UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

AUTOR: LUIS ARENAS HERNÁNDEZ

Informática III (Estructuras de datos estáticas y dinámicas en memoria secundarias)		Clave: 1367
Plan: 2005		Créditos: 8
Licenciatura: Informática		Semestre: 3°
Área: Informática(desarrollo de sistemas)		Hrs. Asesoría: 4
Requisitos: Informática II (Estructura de datos estáticas y dinámicas en memoria principal), 2º. Semestre.		Hrs. Por semana: 4
Tipo de asignatura:	Obligatoria (x)	Optativa ( )

Objetivo general de la asignatura

Al finalizar el curso, el alumno será capaz de implantar y manipular las estructuras de datos estáticas y dinámicas en memoria secundaria.

El alumno conocerá y manipulará los algoritmos más comúnmente utilizados de la programación, donde se involucren las Estructuras de Datos y los Archivos.

Temario oficial (64 horas sugeridas).

1. Tipos de archivos de acuerdo a su organización y operaciones sobre estos (20 hrs.)
2. Métodos de clasificación y consideraciones de complejidad (20 hrs.)
3. Métodos de búsqueda (24 hrs.)

Introducción

Un apoyo que tiene el proceso enseñanza-aprendizaje, es precisamente el material didáctico, ya sea a través de apuntes, cuaderno de ejercicios, libros o aplicaciones de computadora, sobre todo si se trata de un área tan dinámica como lo es la de la Informática.

Estos ejercicios pueden ser de apoyo para esta asignatura, pretendiendo dar material para los temas de Archivos, Clasificación (ordenamientos) y Búsquedas, así como de clasificación (ordenamientos) codificados en el lenguaje de programación C y un pequeño análisis comparativo entre estos métodos. Además, se presenta un anexo donde se muestra un método de clasificación (Mezcla) que aunque no lo contempla el temario es de mucha utilidad.

Este apunte es el resultado del trabajo conjunto del autor, que ha impartido esta materia en varios semestres, y de los alumnos, que han cursado esta asignatura. El mejoramiento de esta obra podrá lograrse con la colaboración nuevamente de los alumnos y profesores que la utilicen.

Este trabajo se desarrolla siguiendo los tres grandes temas descritos anteriormente:

ARCHIVOS.- Tipos de archivos de acuerdo a su organización y operaciones sobre estos.
ORDENAMIENTOS.- Métodos de clasificación y consideraciones de complejidad.
BÚSQUEDAS.- Métodos de búsqueda.

La computación en México existe a partir de 1958, actualmente, ya es imprescindible en nuestra sociedad.

Muy pocas actividades, por no decir ninguna, se ha sustraído a la influencia de la computadora, prueba de ello es que si por alguna causa repentinamente esta desapareciera, buena parte de las actividades humanas, como la industria, el comercio, el transporte, la medicina, la banca, las telecomunicaciones, quedarían hundidas en un caos total, tal como sucedió hace algunos años, cuando uno de los satélites de telecomunicaciones del país falló, se paralizaron algunos servicios bancarios y, sobre todo, las comunicaciones de los teléfonos celulares de algunas compañías.

Cada vez con más frecuencia nuestra economía y vida cotidiana está irrevocablemente más ligada a las computadoras y ellas a los seres humanos. Las computadoras son una creación de la humanidad, sin embargo, no piensan, por lo que no cometen errores, solo ejecutan las operaciones que se les indican a través de las instrucciones sobre qué y cómo hacerlo (son los lenguajes de programación), no teniendo más datos que los que se les ha dado o almacenado en su memoria, por lo tanto, no pueden producir un juicio o criterio de selección (salvo, los esfuerzos para simularlo a través de Inteligencia Artificial, pero al fin de cuentas, siguen siendo operaciones e instrucciones dadas por el desarrollador).

Aunado a esto, en los tiempos actuales, el analfabetismo, ya no radica exclusivamente en no saber leer ni escribir, si no también en no saber computación pues casi cualquier actividad que se desarrolle, requiere de alguna forma de programación, mediante una computadora digital.

Es por demás decir que un mal producto de programación, puede traer consecuencias a veces impredecibles. Por lo que es necesario, desarrollar esta actividad con herramientas para el desarrollo de algoritmos como lo son las Estructuras de Datos.

Es Informática III, la segunda parte de estas Estructuras de Datos, la primera parte es Informática II.

A continuación se desarrollan los contenidos de cada uno de los tres capítulos que componen este material.

TEMA 1. TIPOS DE ARCHIVOS DE ACUERDO A SU ORGANIZACIÓN Y OPERACIONES SOBRE ESTOS

Objetivo particular.

Al finalizar este capítulo el alumno:

Identificará lo que son los conceptos de archivo y el Sistema de Archivos, también conocerá y comprenderá sus organizaciones básicas y las operaciones más comunes que se pueden hacer sobre ellos.

Temario detallado

1.1. Presentación de datos

- 1.1.1. Estructura Jerárquica
  - 1.1.1.1. Campos
  - 1.1.1.2. Registros
  - 1.1.1.3. Archivos
  - 1.1.1.4. Base de Datos

1.2. Terminología

- 1.2.1. Clave
- 1.2.2. Registro Físico y Bloque
- 1.2.3. Factor de Bloqueo
- 1.2.4. Tipos de Datos, Estructuras de Datos y Tipos Abstractos de Datos

1.3. Organización de archivos

- 1.3.1. Secuenciales
- 1.3.2. Directos
- 1.3.3. Secuencial-Indexada

1.4. Operaciones sobre archivos

- 1.4.1. Creación, consulta y actualización
- 1.4.2. Clasificación, Reorganización
- 1.4.3. Destrucción
- 1.4.4. Fusión
- 1.4.5. Rotura/Estallido

1.5. Mantenimiento de archivos
1.6. Algoritmos para manipular archivos secuenciales
1.7. Algoritmos para manipular archivos indexados
1.8. Tratamientos de colisiones
1.9. Acceso a archivos directos mediante indexación

Introducción

El almacenamiento de datos en variables y arreglos se realiza en memoria principal, eso quiere decir que son volátiles por la naturaleza misma de este tipo de memoria y, cuando se termina de ejecutar el programa, todos esos datos se pierden. Para la conservación permanente de datos se utilizan los archivos, los cuales hacen uso de la memoria secundaria.

El hecho de hablar sobre sistemas de cómputo o de información, lleva a relacionarlo con lo que es un archivo, por lo cual se comenzará por definir qué es un **archivo**: es un conjunto de uno o varios registros semejantes y son una colección de datos relacionados entre sí, que pueden utilizarse de una misma forma, localizada o almacenada como una unidad en la memoria secundaria de la computadora. Sirve para entradas y salidas de la computadora digital y son manipulados por los programas. Para dicha manipulación juega un papel muy importante una marca que se conoce como **EOF** (*End Of File*), que indica el fin del archivo.
Los archivos, también se pueden encontrar bajo el nombre de ficheros o files, dependiendo de la fuente de consulta.

Para el manejo de los archivos se tendrán que precisar algunos conceptos como los que a continuación se describen:

Datos e información

Desde la prehistoria, la humanidad ha tenido la necesidad de generar y usar datos ya sea para registrar hechos, para el comercio o para el avance de las ciencias. Ahora, en la época moderna, con el auge de las computadoras se ha

intensificado aún más el uso de los **datos**, ya que lo que una computadora hace es, precisamente, manipular y manejar estos datos que son proporcionados a través de los dispositivos de entrada y, una vez que son tratados de acuerdo a los algoritmos de los programas, son transformados en información y desplegados por medio de los dispositivos de salida o almacenados en memoria secundaria.

La importancia de contar con información correcta, oportuna, en el tiempo adecuado y con un costo razonable, es, precisamente, para poder ayudar a una buena toma de decisiones dentro de la organización a donde pertenezca esta información.

Nivel lógico y físico

Para poder estudiar y manejar los archivos existen dos niveles de clasificación: el físico y el lógico.

El **nivel físico** se refiere a cómo está almacenado físicamente el archivo y dependerá precisamente del dispositivo físico, ya que cada uno manejará sus propias características (cintas o discos magnéticos por ejemplo).

El **nivel lógico** se refiere a cómo se va a manipular por parte del usuario a través de sus programas de aplicación, independientemente del sector, pista, superficie (en caso de disco magnético) o posición de la cinta en que se encuentre.

Sistema de archivos (interfaz)

Para poder manejar estos dos niveles de archivos, se requiere de una interfaz conocida como Sistema de Archivos, que tiene por función permitir al usuario manejar sus archivos independientemente de la forma en que estos se almacenan físicamente en la memoria secundaria. El Sistema de Archivos forma parte de los programas del Sistema Operativo y no es igual a un manejador de Bases de Datos, ya que su función es sólo dar la traducción

entre los dos niveles (físico y lógico) y, dependiendo de su complejidad, permitirá tener diferentes tipos de organizaciones de archivos, que más adelante se verán.

Administración de datos

Para que un Sistema de Información sea útil, es necesario que éste funcione bien, pero sobre todo que los datos con los que se esté alimentando sean confiables y correctos, ya que por muy bien que funcione el sistema de información, si los datos con que se alimenta son incorrectos, nos dará una información errónea, por lo que se recomienda tener sumo cuidado en las siguientes etapas de su administración:

- Captura.
- Validación.
- Clasificación.
- Almacenamiento.
- Recuperación.
- Protección.

A continuación se presentan algunas de las características de los archivos:

- Residencia en soportes externos que es la memoria secundaria o masiva. Independencia de la información o datos con respecto de los programas de aplicación.
- Permanencia en cuanto al tiempo de existencia del archivo, ya que no se limita solamente al tiempo de ejecución del programa que lo crea, si no que permanece por tiempo indefinido, hasta que se borre explícitamente mediante la instrucción de un programa o por el sistema operativo.
- Gran capacidad de almacenamiento, casi ilimitada.
- Un archivo puede ser accedido por distintos programas en distintos momentos.

Entre las **ventajas** de la utilización de **archivos** se encuentran:

- La mejor portabilidad de los datos de una computadora a otra.
- Un mayor volumen de datos, que se pueden almacenar.
- Un mejor acceso a dichos datos, ya que permite varios tipos de acceso.

- Permitir compartir estos datos por varios programas o aplicaciones.
- Tener un mayor grado de confianza en el contenido de estos archivos, ya que si una aplicación modifica alguno de los datos, este se realiza y se ve reflejado cuando alguna otra aplicación, la consulte**.**

Entre las **desventajas** se encuentran:

- Una mayor inversión de tiempo y de trabajo para la realización de aplicaciones utilizando archivos comparados con aplicaciones de *SQL (Lenguaje estructurado de consultas)*, de los manejadores de bases de datos.
- No todas las organizaciones de archivos permiten realizar búsquedas en forma eficiente o con técnicas como la transformación de llaves, como es el caso de la organización Secuencial.

Los sistemas de archivos no siempre tienen todas las organizaciones.
Algunas organizaciones como la secuencial no permiten la actualización en línea (*on line*), es más enfocada a la actualización en lote (*batch*).

1.1. Presentación de datos

Los datos son de diferentes tipos, pueden ser primitivos o compuestos. Los tipos de datos primitivos, también conocidos como elementales, ya viene implementados en los lenguajes de programación de tercera y cuarta generación, por lo que no es necesario implementar algún algoritmo, para manipularlos entre los más comunes se tienen: los enteros, enteros sin signo, los reales, los carácter y los arreglos. Siendo el carácter quien se encuentra en el nivel mas bajo en la estructura jerárquica de las Estructuras de Datos.
Los tipos de datos compuestos son los que generalmente si requieren de implementar algún algoritmo para manejarlos, como ejemplo de estos datos se tienen las listas en sus diferentes formas, como son las Pilas, Colas, Colas circulares y Colas dobles.

1.1.1. Estructura Jerárquica

Los conceptos de carácter, campo, registro, archivo y base de datos son conceptos referentes a la organización lógica (forma en que el usuario de una computadora digital manipula o ve a los datos), las Estructuras de Datos, se pueden organizar de forma jerárquica, como si se tratara de un árbol, donde el nivel de mayor jerarquía es la Base de Datos y el de menor es el carácter.

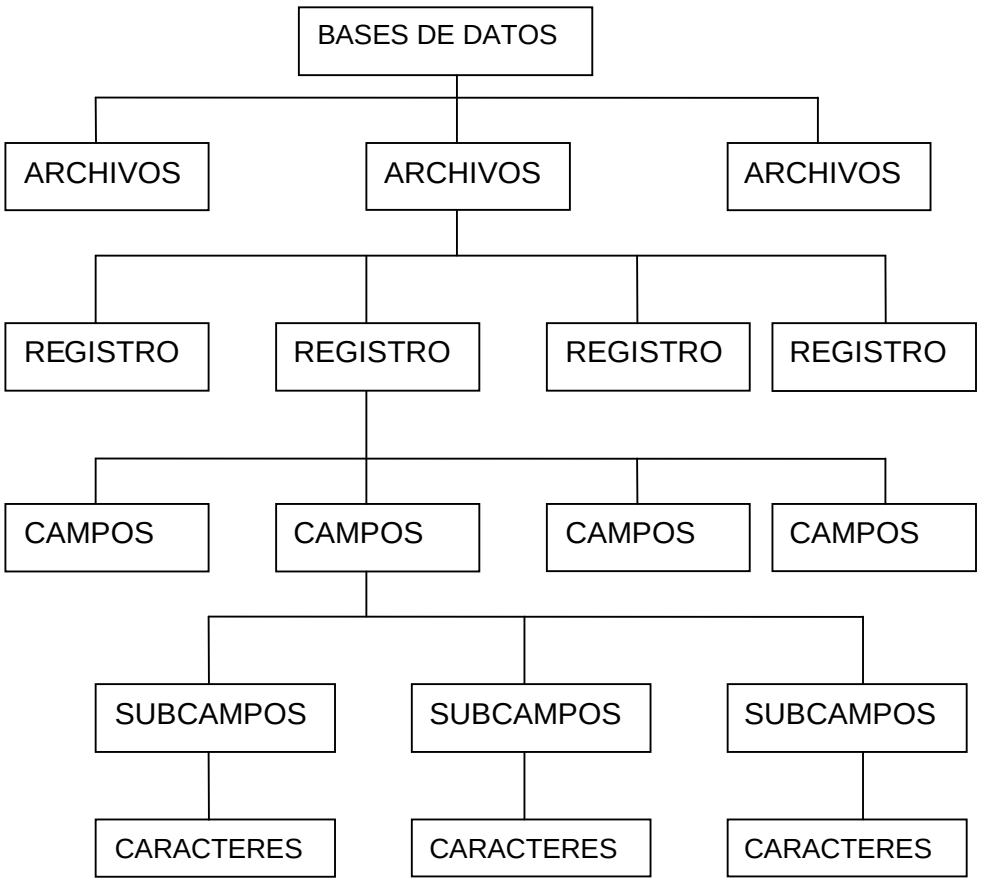


Figura 1.1. Estructura jerárquica

Para poder realizar el estudio de los archivos, es necesario puntualizar algunos de estos conceptos, así como su relación jerárquica, como los que a continuación se describen:

1.1.1.1. Campos

Los datos son representados por números, letras y caracteres especiales y son identificados por un ente (**entidad**) llamado campo, el cual es un grupo de caracteres, por ejemplo el nombre del alumno, su número de cuenta, clave de carrera, etc.

Los campos que conforman un registro pueden ser de diferente tipo y longitud. Por ejemplo, Nombre del alumno (longitud de 30 caracteres alfanuméricos), número de cuenta (longitud de 8 caracteres numéricos).

Existe lo que se conoce como **articulo elemental** (algunos autores lo denominan subcampo), que es aquel subcampo que no se puede descomponer en más campos y el articulo de grupo (o campo), el cual, sí se puede descomponer en más campos, como por ejemplo el RFC (registro federal de contribuyentes) que se puede descomponer en las iniciales del nombre y del apellido, así como en la fecha de nacimiento.

Generalmente, los campos y subcampos son de longitud fija, pero algunos Sistemas de Archivos pueden manipularlos de longitud variables, aunque son muy pocos los que lo hacen. Por ejemplo si se capturan los nombres de los títulos de libros para la Biblioteca, estos pueden ser de longitud variable.

1.1.1.2. Registros

Un conjunto finito de campos, que pueden ser de diferente tipo y tienen una relación lógica conforman un registro, el cual es utilizado como una unidad. Un ejemplo sería, el registro de un alumno, que es conformado por campos como los siguientes:
Número de cuenta, Nombre, Clave de la carrera, Clave de la Facultad, etc.

Al igual que los campos, también pueden ser de longitud fija y de longitud variable, siendo esta última la menos común.

Por ejemplo retomando el caso de los libros para la Biblioteca, si además de los títulos de los libros se capturarán los títulos de los capítulos, entonces no tan solo se tendrían campos de longitud variable, si no que también números de campos variables, ya que los libros no tienen un número fijo de capítulos.

1.1.1.3. Archivos

Como se mencionó anteriormente, un **archivo** es un conjunto de uno o varios registros semejantes, los cuales pueden ser de longitud fija o variable, y, a su vez, cada registro es un conjunto de uno o varios campos relacionados que contienen a los datos elementales, los cuales también pueden ser de longitud fija o variable, e incluso, el número de campos puede ser fijo o variable. Estos registros se conservan en dispositivos de computadora de almacenamiento secundario como pueden ser discos, cintas magnéticas y, más recientemente, memorias USB.

El archivo de una computadora también se puede definir como una estructura diseñada para guardar datos, con el objetivo de poder ser recuperados o almacenados fácilmente.

1.1.1.4. Base de Datos

Este campo del conocimiento ha evolucionado vertiginosamente en los últimos años gracias al avance de la electrónica y de las telecomunicaciones, lo que ha permitido llegar a tener grandes cantidades de datos, dando auge a conceptos como los de Minería de Datos.

Una Base de Datos se puede definir de varias formas:
"Es una colección de archivos a los que puede accederse por un conjunto de programas y que contienen todos ellos datos relacionados"[1]

[1] Apuntes de Informática III, p.111.

Según la enciclopedia virtual Wikipedia,

"una base o banco de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta"[2].

De acuerdo al sitio de Internet monografías, es

"un conjunto de información almacenada en memoria auxiliar que permite acceso directo y un conjunto de programas que manipulan esos datos"[3].

Otra definición que tal vez sea la más general, menciona que:

Una Base de Datos es una colección de información que se almacena por un periodo largo de tiempo, que puede llegar a ser de años (sobre todo ahora está en auge la Minería de Datos, que se basa en parte en este principio, de tener información por periodos largos de tiempo, para poder tener mas elementos en la toma de decisiones) y que es manejado por un Sistema de Administración de Base de Datos conocido como *Data Base Management System* (DBMS) o simplemente sistema de bases de datos.

Entre las **características** que se esperan del **funcionamiento de las bases de datos** a través del DBMS están el permitir a los usuarios:

- Crear tanto las bases de datos como su esquema, el cual es su estructura lógica de datos y esto lo logra a través del uso de lenguajes de definición de datos.
- Consultar para la realización de: altas, bajas, cambios y reportes a las Bases de Datos, a través de un lenguaje de consulta *SQL (Structured Query Language).*
- Almacenar grandes cantidades de datos, llegando a Gyga bytes y ya se está contemplando Tera bytes. Con lo que se están desarrollando lo que en la

[2] http://es.wikipedia.org/wiki/Base_de_datos

[3] http://www.monografias.com/trabajos11/basda/basda.shtml

actualidad se conoce como **Minería de Datos**, en la cual se trata de adelantarse a las necesidades de información de los usuarios, con el análisis de grandes volúmenes de información, que puedan ser fácilmente consultados.

- Almacenar los datos por grandes periodos de tiempo, implantando medidas de seguridad contra accidentes o permitiendo el acceso sólo a usuarios o aplicaciones autorizadas.
- Controlar el acceso simultáneo por varias aplicaciones, sin que las acciones que realice alguna de ellas afecten a las demás.

Una diferencia fundamental entre un Sistema de Administración de Base de Datos (DBMS) y un Sistema de Archivos, es que el DBMS utiliza al Sistema de Archivos, para realizar sus tareas.

En general, los componentes de una **Base de Datos** son los tres que se presentan a continuación:

Usuario o programa de aplicación. Los usuarios interactúan con el sistema administrador de la Base de Datos, ya sea en forma directa a través del lenguaje estructurado de consultas *SQL,* o a través de programas de aplicación escritos en algún lenguaje de tercera o cuarta generación, que ya está pasando a desuso.

Sistema de administración de Base de Datos. Es la interfaz entre los programas de aplicación o los usuarios y los datos que ellos procesan, que fueron descritas sus características en el apartado 1.1.

Base de Datos. Es la colección de datos altamente estructurados, que son una especie de tablas o matrices, donde aparecen renglones que son los registros y las columnas son los campos.

Las bases de datos empezaron a tener presencia en la década de los sesentas y con el paso de los años, se crearon tres **modelos principales**, que fueron:

El jerárquico que utiliza una estructura de árbol, para conectar a sus registros.

El de red donde sus registros se interrelacionan unos con otros formando precisamente una red

El relacional, en el cual las tablas de los registros se relacionan unas con otras a través de campos comunes, siendo este modelo el que ha evolucionado y prevalecido más que los otros y en la actualidad es el más popular y usado.

1.2. Terminología

1.2.1. Clave

Generalmente, existe un campo dentro del registro al que se le denomina campo clave o llave, el cual es un campo de identificación único del registro, esto quiere decir que su contenido no se repite, por ejemplo, el número de cuenta de un alumno que aparece en todos los registros. Sin embargo, pueden existir archivos que no contengan esta llave, como sería el caso de archivos bibliográficos, donde se capturarán los títulos y autores de cada capítulo o de todo el libro, pudiendo inclusive repetirse algunos de ellos.

1.2.2. Registro físico y bloque*

Antes de definir lo que son estos dos conceptos, convendría precisar lo que es un **registro lógico**, el cual se puede definir como el número de caracteres, palabras o campos que se obtienen al ejecutar las instrucciones de lectura o escritura en un programa de aplicación. En otras palabras, es la forma cómo el usuario o su programa de aplicación manipulan los registros.

Registro Físico o Bloque es un grupo de registros lógicos, almacenados física y contiguamente en disco y es la cantidad mínima de información que se transfiere en cada operación de lectura o escritura que se hace sobre un archivo, por ejemplo, una línea de impresión o un sector de un disco, etc.

1.2.3. Factor de bloqueo

Es el número de registros lógicos contenidos en cada bloque. Se da por la relación del tamaño del **registro físico** entre el tamaño del **registro lógico**. A

* Estos dos conceptos son equivalentes.

mayor tamaño del **factor + de bloqueo**, se realizan menos operaciones de entradas y salidas al disco, pero también implica que el tamaño de la memoria principal se vea reducido al darle mayor espacio a este **factor de bloqueo**.

1.2.4. Tipos de datos, estructuras de datos y tipos abstractos de datos

Tipos de Datos

Se puede definir un tipo de dato a partir de los valores permitidos que puede tomar éste y las operaciones que se puedan llevar a cabo sobre estos valores.

"Se dice que un lenguaje dispone de tipado[4] fuerte si es posible determinar en tiempo de compilación si las operaciones que se realizan sobre los datos son consistentes con el tipo de los mismos.
Fortran, Cobol o C no disponen de tipado fuerte, mientras que Ada, Java o C++ sí están fuertemente tipados". [5]

Se pueden clasificar en **tipos de datos:** elementales o primitivos y compuestos, que como se mencionó en el apartado 1.1. los datos elementales ya vienen implementados en los lenguajes de programación a partir de la tercera generación, tales como los enteros, los reales, los arreglos y los de tipo carácter; los tipos compuestos generalmente hay que manejarlos a través de algún algoritmo, como es el caso de las listas, las cuales son tratadas a detalle en la asignatura de Informática II.

A la hora de analizar, que lenguaje de programación se va a utilizar se consideran algunos aspectos como los que a continuación se describen:

---

[4] Se refiere a los diferentes tipos de datos que se pueden manejar en un lenguaje de programación, tales como los descritos en el apartado 1.1. Algunos lenguajes manejan una mayor cantidad de tipos de datos.

[5] http://www.dte.upct.es/personal/balvarez/Docencia/Fundamentos/Tema_2.pdf

El número de tipos primitivos que maneja.
Ver si permite mezclar los tipos de datos primitivos para crear otros tipos de datos más complejos como son las estructuras.
Contemplar si pueden realizarse *conversiones de tipos* de forma segura (*casting*) como el cambiar de tipo entero a carácter, en el caso de imprimir gráficas.

Tipos Abstractos de Datos (TDA)

Cuando se definen los algoritmos y las Estructuras de Datos a partir de la función de las operaciones efectuadas y no en los detalles de la implementación se conoce como: Tipos de Datos Abstractos. Esto es enfocarse más a qué es lo que hace y no en el ¿cómo lo hace?
Una de las características de este tipo de **datos abstracto** es que nada que sea externo a la definición de las estructuras de datos y a los algoritmos que operan sobre estas estructuras debe hacer referencia a cualquier cosa interna, salvo que se realice a través de llamadas a funciones o procedimientos de las operaciones fundamentales de éstas.
Se han desarrollado este Tipos Abstractos de Datos para facilitar la actividad del desarrollador en el momento de realizar sus programas, ya que permiten organizar grandes programas, porque reduce el tamaño y la complejidad de la interfaz entre algoritmos.

Un ejemplo claro de este tipo de datos, son los diferentes tipos de listas, como bien podrían ser las Pilas[6], en la que interesa más el hacer el *PUSH (*Agregar un elemento) y el *POP (*retirar un elemento), que en ver como lo hace.

Cuando empezaron las primeras generaciones de los lenguajes de programación, no existían este tipo de datos, pero después se fue notando que varias actividades se repetían, como la del *PUSH* y del *POP*, mencionadas

[6] Las listas como Pilas, Colas, Colas Circulares y Colas Dobles, se tratan más a detalle en la asignatura de Informática II y son Estructuras de Datos Compuestas.
Siendo las Pilas las que manejan la políticas de Últimas Entradas, Primeras Salidas.

anteriormente, como ejemplo tenemos el caso del lenguaje de programación C, que se tienen en las librerías TDA, al ser lecturas y escrituras, o funciones trigonométricas y el manejo de caracteres, en lugar de estar repitiendo código de programación, mejor se deja uno sólo que puede ser utilizado por el programa de aplicación que lo desee.

1.3. Organización de archivos

Clasificación de archivos

Existen diferentes formas para clasificar archivos de acuerdo a los siguientes criterios:

- Por la forma de acceso

Al utilizar un archivo, es necesario definir el tipo de acceso. Esto se logra cuando se abre un archivo a través de un lenguaje de programación y se tienen las siguientes formas:

Entrada.

Salida.

Entrada/salida.

- Por su forma de organización

La organización de un archivo es la forma en la que los registros se acomodan en los dispositivos de almacenamiento, o bien cómo se estructuran los datos dentro de un archivo.

Se tienen en forma general las siguientes cuatro organizaciones:

- Secuencial.
- Directa.
- Indexada (o secuencial con índice).
- Con varias llaves (menos frecuente).

1.3.1. Secuenciales

Son aquellos en que los registros están escritos, uno a continuación de otro y para poder acceder a un registro **n**, es necesario haber recorrido los **n-1** registros anteriores, por poner una analogía, se pueden comparar con un cassette de música, en donde si se quiere escuchar la canción número cinco, va a ser necesario haber escuchado las cuatro melodías anteriores (o hacer el corrimiento de la cinta del cassette hasta llegar a la canción cinco).

También se puede decir que es un conjunto de **registros almacenados consecutivamente**. Es importante mencionar que tanto leer, como escribir, se realizan en forma consecutiva. Algunas de las características que tienen los archivos secuenciales son:

La escritura de nuevos datos, siempre se realiza al final del archivo, o bien después del último registro, si es que ya tiene algunos.

Para leer una porción del archivo en particular, hay que seguir adelante en el archivo siempre, si ya estuviera antes del registro actual, se tendrá que "rebobinar", esto es regresar al inicio o primer registro del archivo.

Sólo se pueden abrir para lectura o para escritura, pero nunca las dos operaciones al mismo tiempo.

Este tipo de organización es muy utilizada para la actualización de datos cuando se realiza el **proceso en lote** (*batch*), ya que permite acceder al siguiente registro rápidamente. Un **proceso en lote** es aquel en que la actualización no se hace instantáneamente de los datos, si no en cada determinado periodo de tiempo que es cuando se ejecutan los procesos que realizan esta actualización.

El desempeño de los archivos secuenciales depende principalmente de:

- El factor de bloqueo.
- La longitud del archivo
- Selección de la llave.

Este tipo de archivos, es muy común aunque en el pasado lo era más, ya que por mucho tiempo se ocupaban principalmente **procesos en lote**, sobre todo porque era muy difundido el uso de las tarjetas perforadas, las cuales son el caso típico de un archivo secuencial.

Otro ejemplo donde se utilizan archivos secuenciales es en las cintas magnéticas que cada vez van siendo menos usadas, pero también se pueden utilizar este tipo de archivos en disco.

Sólo permiten las operaciones de lectura y escritura, una a la vez, no simultáneamente.

A continuación, en la figura 1.2. se muestra el tipo de organización secuencial

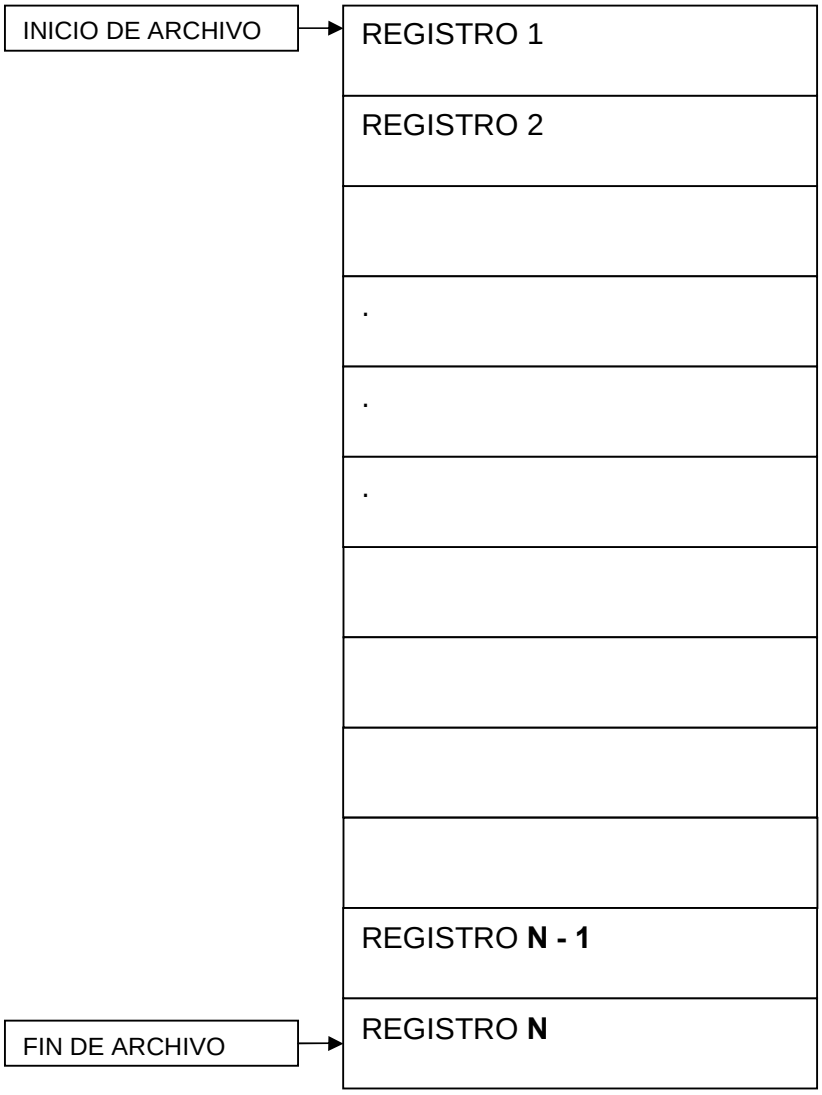


Figura 1.2. Organización Secuencial

1.3.2. Directos (Aleatoria)

Cada registro puede leerse y/o escribirse de forma directa, basta con indicar la dirección donde se desea colocar y/o leer un registro en particular dentro del archivo, ya sea través del número relativo del registro o por transformaciones que se realizan a la clave (llave) de dicho registro.

El orden físico en el que se fue grabando cada uno de los registros de este tipo de archivos, pueden o no corresponder al orden lógico con que son tratados.
En la figura 1.3. se ejemplifica esta idea:

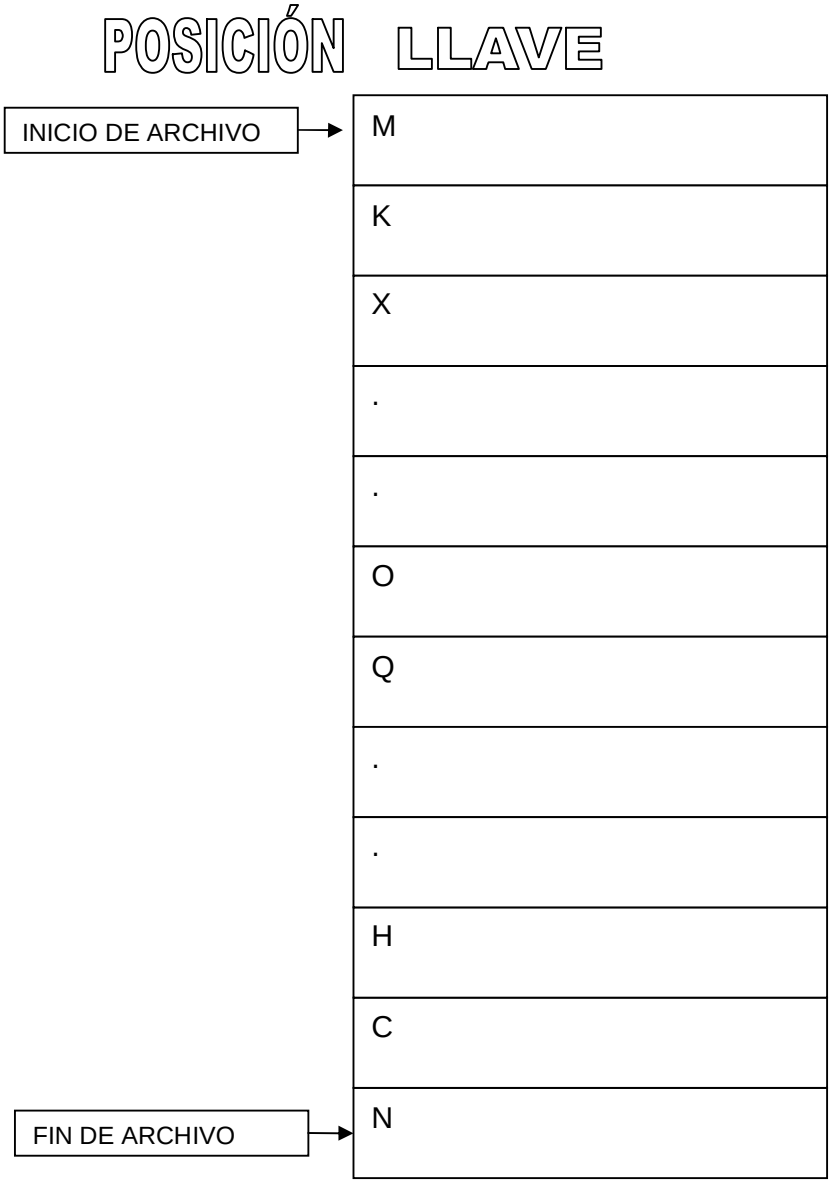


Figura 1.3. Organización Directa (Aleatoria)

Los registros no aparecen físicamente en orden por el valor de la llave, es importante hacer notar que se puede acceder a los registros del archivo en el orden consecutivo físico en que fueron llegando, pero es posible que el valor de las llaves no se encuentre en la misma secuencia lógica. Para el ejemplo de la figura anterior, si el acceso a los registros fuera secuencial, los registros se recuperaran en el siguiente orden:

M, K, X,..O, Q,..H, C, N.

Son muy rápidos en cuanto al acceso de la información que contienen ya que no se tienen que recorrer todos los registros que estén antes de él, se va directamente a la posición **n** del registro buscado.

Esto se logra con el uso de una relación, que tiene como entradas los valores de una llave y da una dirección como salida (también se conoce como **direccionamiento**). Es importante recalcar que la relación que se utilice para almacenar, sea la misma que se ocupe para recuperar los valores de las llaves.

Para poder hacer esto existen las varias alternativas, entre las más comunes se encuentran las siguientes:

1) Mapeo directo.
2) Directorio.
3) Cálculo.

1) Mapeo Directo

Consiste en la conversión entre la dirección, que maneja el usuario y en la que se deja físicamente el registro dentro del archivo.

Esta alternativa tiene dos caminos:

- Direccionamiento absoluto.
- Direccionamiento relativo.

- Direccionamiento absoluto

Este método es uno de los más rápidos ya que se utiliza el valor de la llave como la dirección física, por lo que no se necesita tiempo de procesamiento en el cálculo de la dirección para localizar la posición del registro, ya que indica en que dirección física se va a localizar

A pesar de ser muy rápido este método, también tiene sus desventajas, entre las que se encuentran:

La gran dependencia del dispositivo de almacenamiento y del espacio a utilizar, por lo que si se desea emigrar a otro equipo de cómputo con dispositivos de almacenamiento diferentes, implicará que las direcciones físicas también cambien. Puesto que los dispositivos de almacenamiento para este tipo de organización son los discos y para localizar físicamente un registro se requiere del número de la pista, del sector y de la superficie.

Es que la llave a veces no se podría hacer que coincida con posiciones reales, por ejemplo si se trata de campos no numéricos como el **R**egistro **F**ederal de **C**ontribuyentes o a pesar de ser numéricos, maneje rangos grandes como un número de cuenta de una tarjeta bancaria de dieciséis dígitos, lo cual haría de gran tamaño el archivo en cuanto a número de registros y quizás no todos se llegaran a ocupar.

- Direccionamiento relativo

En este tipo de direccionamiento, la dirección de un registro en el archivo es el número del registro.

Esta dirección debe de ser un número entero. Un archivo de tamaño **N** de registros, tendrá un rango de direcciones relativas para los registros que almacenará entre **1 y N** (hay algunos manejadores de Archivos, que empiezan en la dirección cero y entonces el rango quedaría entre **0 y N-1**).
La función de mapeo directo tendrá como entrada y salida la misma llave, pero indicando en qué dirección relativa dentro del archivo se va a encontrar. Por ejemplo, si se trata del registro cuya llave es el cinco, quiere decir que se va a encontrar en la posición cinco del archivo, independientemente de cuál sea su posición física (pista, sector y superficie).

Es importante resaltar, que el tamaño del archivo se calcula por el rango de valores que pueda llegar a tener la llave y no tanto por el valor más alto de la

ésta. Por ejemplo, si se tienen los números de cuenta de los alumnos de una escuela, los cuales constan de cinco dígitos, y cada año al inscribirse, los dos primeros dígitos, indican el año de ingreso entonces, sólo quedarían tres, con los cuales se puede manejar un rango de mil combinaciones (del 000 al 999). Por lo que el tamaño del archivo sería de mil elementos y no del valor más grande de la llave, que sería 99999.

Este tipo de direccionamiento, también es casi tan rápido, como el del direccionamiento absoluto, ya que sólo consume poco tiempo de procesamiento, para el cálculo de la dirección, para cambiar de la dirección relativa a la dirección absoluta, el cual lo realiza el Sistema de Archivos.

Con este tipo de direccionamiento, se pueden manejar diferentes tipos de estructuras de datos, como son las listas simples, doblemente ligadas (o con varias ligas en algunos casos).

2) Directorio

En este tipo de direccionamiento, se utiliza una tabla llamada directorio o diccionario (espacio adicional), el cual va a tener el valor de la llave y la dirección relativa como se muestra en la siguiente figura:

Figura 1.4. Directorio

Para localizar un registro determinado en el archivo de datos, en este método se encontrará primero en el directorio (tiempo adicional) el valor de la llave y posteriormente se tomará la dirección relativa correspondiente a esa llave, para acceder al archivo de datos.

Gran parte del éxito en cuanto a la rapidez de la búsqueda dependerá de la organización del directorio, ya que es muy recomendable que los valores de la llave, estén clasificados en orden ascendente (o descendente) con el fin de utilizar la búsqueda binaria (este tema se profundizará más en "Métodos de Búsqueda" en el tema 3). Tal vez una de las desventajas es que cuando se realice alguna actualización de **altas** o **bajas** de un registro, el Directorio debe de quedar en el mismo orden de clasificación, por lo que se tendrán que realizar corrimientos del resto de los elementos del Directorio, ya sea hacia arriba (si se trata de una **baja**) o hacia abajo (si se trata de una **alta**), lo cual implica tiempo adicional de procesamiento.

Otra forma mejorada para evitar los corrimientos que se tendrían que hacer en los elementos de la tabla que está organizada en forma contigua es cambiarla por una organización simplemente ligada, ahorrando tiempo de procesamiento, pero aumentando el espacio de memoria utilizado, ya que involucraría más por las ligas.

En este método de Directorio, lo que hay que optimizar son los dos factores que se han estado mencionando:
Espacio adicional de memoria.
Tiempo de procesamiento.

En el tema 2, se verá más a detalle alguno de los criterios que se siguen para medir la eficiencia de los algoritmos.

3) Cálculo

Este tipo de direccionamiento se da a través de la manipulación del campo llave a través de la programación entre el contenido de dicha llave y la posición que ocupa dentro del archivo. Esto se hace a través de la utilización de las funciones de Mapeo también conocidas como funciones de Hash.

La función de dispersión es la relación que tiene como entrada el valor de una llave y da como salida la dirección, en la cual se encuentra (por almacenar o por recuperar).

La ejecución de una función de Hash depende de varios factores como:

La **distribución** actual de los valores de las llaves.

El **número de registros** con llaves diferentes que pueden ser almacenados en una dirección dada.

El **número de valores** de las llaves que se estén utilizando con respecto al tamaño de direcciones.

La utilización de las **técnicas para la solución de colisiones**. Puede presentarse el caso de que para una misma dirección existan, de acuerdo a la función de mapeo, más de un registro que deba estar ahí, es lo que se conoce como **colisión**.

Las funciones de H*a*sh serán tratadas un poco más a detalle en el tercer tema de estos apuntes.

Para poder implantar un archivo de organización directa, se tienen que considerar los siguientes puntos:

Que el dispositivo de memoria secundaria pueda ser direccionable en cualquier posición como lo son los discos, ya sea magnéticos (Disco Duro o Diskketttes), o bien de tecnología láser como los CD o DVD.

Cada registro debe de tener un campo llave, para diferenciar un registro de otro.

Una correspondencia entre los valores de la llave y las direcciones disponibles del dispositivo de memoria secundaria.

El acceso en ambos casos implica la existencia de huecos, es decir, direcciones de registros dentro del archivo, que no serán ocupados, recomendándose que sea de aproximadamente del 20% del total de registros que se estime ocupar realmente, sobre todo si se implementa a través de la segunda opción mencionada anteriormente, con el fin de que si se llegan a presentar colisiones tener espacios libres en memoria, para reducir sus efectos y resolverlas rápidamente por algunos de los métodos clásicos, como Direccionamiento Abierto o Encadenamiento. Estos métodos también se tratarán más a detalle en el tercer tema de este trabajo.

Este tipo de archivos, sólo se pueden utilizar en discos debido a sus características físicas de poder acceder a sus registros por las coordenadas de sectores y de pistas.

Por hacer nuevamente una analogía, con un CD de música, si se desea escuchar la canción cinco, no es necesario escuchar las cuatro anteriores, sabiendo la posición que es la cinco, se va directamente a ella, así sucede con este tipo de organización.

LLAVE POSICIÓN

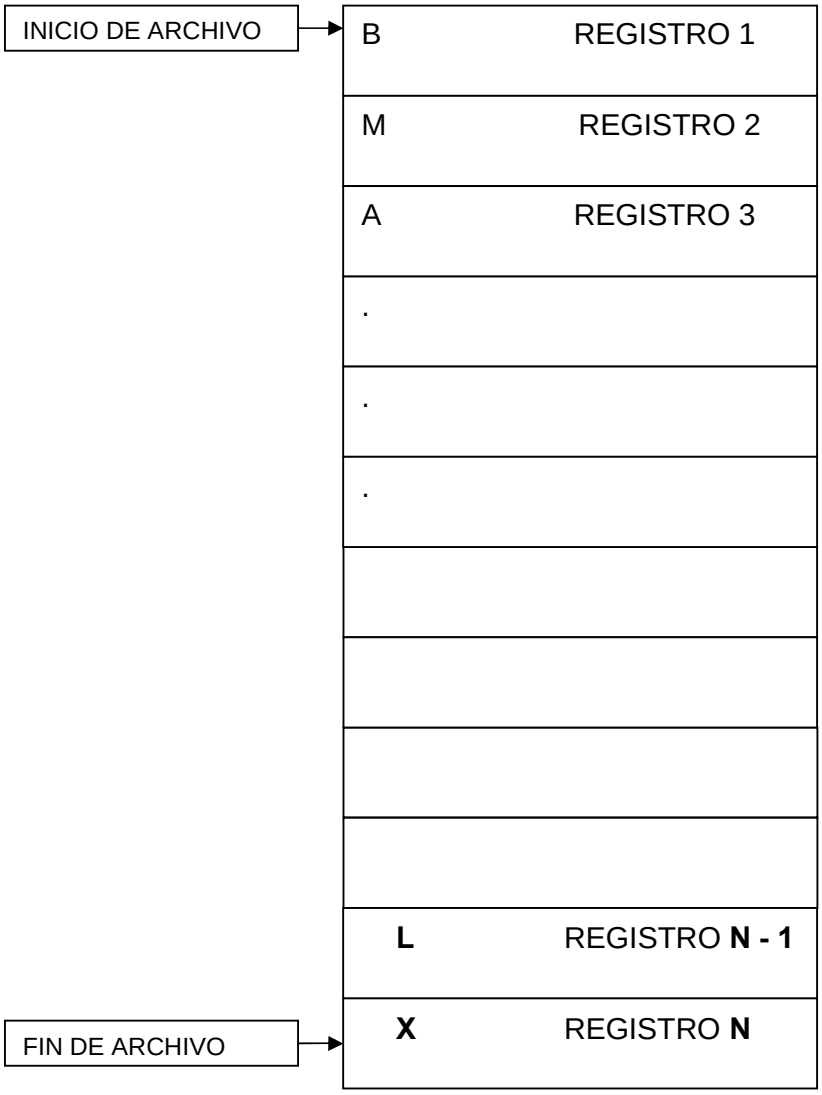


Figura 1.5. Organización Directa

La recuperación secuencial de este archivo sería **B, M, A, ... L, X.**

Por lo general, permiten simultáneamente las operaciones de lectura y escritura, y una a la vez.

1.3.3. Secuencial-Indexada

Esta organización está diseñada para utilizar la combinación de la organización directa y de la secuencial.

A través de esta organización es posible el acceso a un registro en particular (aleatoria) por medio de la llave (la cual es sólo una, la primaria) y también a la organización secuencial, ya sea desde el inicio o desde cualquier otro registro del archivo.

Los registros en este tipo de archivos se almacenan de acuerdo a una secuencia física dada, lo más común es seguir el orden indicado por la llave primaria, con lo que se puede tener un acceso secuencial de los registros, pero también se puede acceder aleatoriamente, ya que se puede llegar a los registros en cualquier orden, no precisamente relacionados con su distribución física.

Para poder realizar este tipo de organización, es necesario contar con tres áreas:
De índices.
Primaria o de datos.
De excedentes o de desbordamiento o de *overflow.*

El **área de índices** es creada por el sistema, en donde cada registro establece una división o segmento en el área primaria y contiene la dirección del comienzo del segmento, así como su clave o llave más alta.

Se entiende el índice como una referencia para obtener de forma automática la ubicación de la zona del archivo físico, donde se encuentra el registro que se está buscado, con lo cual se puede localizar un registro por medio de su llave sin tener que recorrer previamente todos lo que le preceden. El archivo de índices utiliza como entrada la llave primaria y da como salida una información referente a la ubicación física del registro.

El **área primaria** tendrá los registros de datos, los cuales estarán clasificados en forma ascendente de acuerdo a su campo llave.

El **área de excedentes**, sirve para agregar nuevos registros que ya no son colocados en el área primaria al actualizar el archivo

En la estructura secuencial indexada más sencilla, se usa un nivel de indexación. El índice, en este caso, es un archivo secuencial simple. Cada registro del archivo índice tiene dos campos: **un campo clave**, que es el mismo que el campo clave del archivo principal y **un apuntador** que va al archivo principal.

Para **localizar** un **campo específico** se realizan los siguientes pasos:

- Se busca en el índice hasta encontrar el valor mayor de la clave que es igual o precede al valor deseado de la clave. La búsqueda continúa en el archivo principal a partir de la posición indicada por el apuntador.
- Los registros del archivo principal cuentan con un campo adicional constituido por un apuntador al archivo de desbordamiento (*overflow*). Cuando es insertado un registro nuevo al archivo, también se añade al archivo de desbordamiento. El registro del archivo principal que precede inmediatamente al nuevo registro, según la secuencia lógica, se actualiza con un apuntador del registro nuevo en el archivo de desbordamiento, si el registro inmediatamente anterior está también en el archivo de desbordamiento, entonces se actualizará el apuntador en el registro nuevo.

- En caso de acceder secuencialmente un archivo completo, los registros del archivo principal se procesarán en secuencia hasta encontrar un apuntador al archivo de desbordamiento, el acceso continúa en el archivo de desbordamiento hasta que encuentra un

apuntador a nulo (*NULL*), esto es, que ya no apunta a ninguna dirección, entonces renueva el acceso donde se abandonó en el archivo principal.

Haciendo otra analogía, es como si se tratará de localizar una palabra en un diccionario, no se busca en todo el diccionario de inicio a fin, primero se va a buscar en el índice (archivo de índices), la palabra (llave) y ahí nos dice en qué pagina empieza esa palabra (apuntador al archivo principal), siendo esta página donde se empieza a buscar, pero sólo a partir de esa dirección y no en todo el diccionario.

Este tipo de organización es muy utilizada, en procesos en línea (*online*), ya que sus búsquedas son bastantes eficientes y rápidas, y es en este tipo de procesos precisamente que la actualización de los datos es prácticamente instantánea, a diferencia del procesamiento por lote, donde hay que esperar intervalos de tiempo para actualizar los datos.

La actualización de los índices: cuando se insertan y eliminan registros, es preciso actualizar los índices para evitar contratiempos actualizando un archivo.

Nótese que no todos los lenguajes de programación soportan esta organización. A continuación se describen algunas ventajas y desventajas de ésta:

Ventajas	Desventajas

Rápido acceso	Desperdicio de espacio dentro del archivo, ya que quedan huecos libres, al ser actualizado el archivo
El Sistema de archivos, es el que se encarga de relacionar la posición de cada registro con su contenido a través de su tabla de índices	Se requiere de más espacio, para el área de índices y de excedentes

Cuadro 1.1. Ventajas y desventajas de la actualización de los índices en la organización Secuencial-Indexada

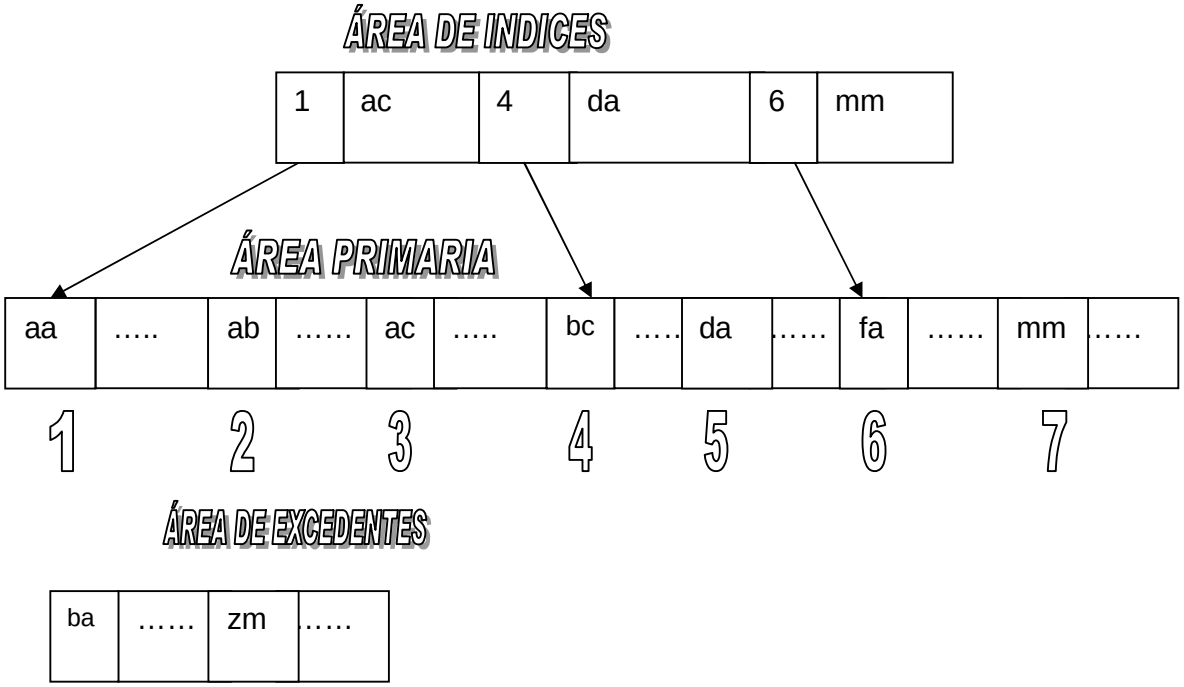


Figura 1.6. Organización Secuencial Indexada

Para este ejemplo el área de índices indica el segmento, que empieza en la dirección 1 del área primaria y su llave de mayor valor o más alta es **ac,** cuya

dirección es el registro 3, en tanto que el siguiente segmento empieza en la dirección 4 y su llave más alta es **da** y el tercer segmento empieza en la dirección 6 y su llave más alta es **mm** y así sucesivamente.

En este tipo de organización se pueden dar las operaciones de lectura y de escritura, una u otra o las dos simultáneamente.

1.4. Operaciones sobre archivos

Existen diferentes formas de operar los registros de un archivo, entre las que se encuentran las siguientes:

Creación	Reorganización
Consulta	Destrucción
Actualización	Fusión
Clasificación	Rotura/Estallido

1.4.1. Creación consulta y actualización

Creación

Al igual que un ser vivo los archivos, tienen un inicio una vida útil y un fin, la creación es la primera operación que se tiene, ya que para poderlo utilizar es necesario que exista.

Para crearlo, generalmente, se hace a través de instrucciones en un lenguaje de programación (que puede ser de diferentes generaciones) y en algunas ocasiones por alguna utilería, del sistema operativo.

En la creación, se define el entorno que permita una eficiente operación del archivo.

Durante el proceso de creación, es necesario, definir, varios parámetros que se conocen como atributos de archivos, entre los que destacan:

- Organización

- Dispositivo al que se direcciona (impresora, disco, Monitor por ejemplo)
- Tamaño del registro lógico (MAXRECSIZE)
- Tamaño del registro físico (BLOCKSIZE)
- Número de áreas (AREAS)
- Tamaño del área (AREASIZE)

Nótese que existen una gran cantidad de atributos de archivos y los nombres que aparecen entre paréntesis pueden llagar a cambiar, así como algunos atributos, dependiendo del sistema de archivos, que se esté utilizando.

Ejemplo

Se tiene un archivo con las siguientes características:

Puede llegar a tener 10,000 registros lógicos.

MAXRECSIZE de 20, ya que cada registro lógico tiene una longitud de 20 caracteres (*bytes*).

Un BLOCKSIZE de 100 caracteres, que es el tamaño del registro físico.

Y AREASIZE de 20 registros lógicos.

Se desea encontrar:

- El Factor de Bloqueo (FB).
- El número de áreas que se necesitan, para almacenar los 10,000 registros físicos.

Solución:

El Factor de Bloqueo indica, cuántos registros lógicos hay en un registro físico. De los datos anteriores, se tiene que el registro físico es de 100 caracteres y el registro lógico es de 20 caracteres, entonces el Factor de Bloqueo es:

FB = BLOCKSIZE/MAXRECSIZE

FB = 100 / 20

FB = 5 registros lógicos en un registro físico.

Si el tamaño de cada área es de 20 registros lógicos, entonces para almacenar 10,000 registros lógicos se necesita realizar la siguiente división para saber cuántas áreas se van a ocupar:

Áreas = 10,000 / 20
Áreas = 500

Se recomienda que al dar los valores de los atributos de archivos, considere, que el tamaño del BLOCKSIZE sea múltiplo del MAXRECSIZE, y que el AREASIZE sea también múltiplo, tanto del BLOCKSIZE como del número de registros lógicos. Esto es con el fin de evitar el desperdicio.

El **registro lógico** es como lo manipula el usuario, el registro físico debe de contener por lo menos un registro físico, al especificar su tamaño, que se transferirá entre la memoria principal y la secundaria.

El tamaño del área debe ser tal que contenga por lo menos un registro físico.

También es importante decir que el Sistema de Archivos define los parámetros con valores predeterminados, los que considera óptimos para él, pero probablemente no lo sean para la aplicación, ya que el usuario es el que conoce mejor las características de su archivo; por ejemplo cuántos registros lógicos va a contener o por lo menos un estimado, y de ahí puede dividirlo en las áreas que él crea conveniente para el crecimiento de sus registros y posteriormente el tamaño de cada área.

Un archivo puede ser creado en un dispositivo, o ser copiado de otro existente en el mismo o diferente dispositivo o bien como el resultado de la mezcla o actualización de otros archivos.

A continuación se hace referencia a un ejemplo de la creación de archivos.

Un **archivo secuencial**, si se trata de datos, se puede crear de dos formas:

1. A través de un editor del equipo de computo, como podría ser Word, bloc de notas o Vi, etc.
2. A través de un programa desarrollado en algún lenguaje de programación, en el cual se puedan leer los datos a partir del teclado o de algún otro dispositivo, para almacenarlos en un archivo.

En el caso particular del lenguaje de programación **C,** si el archivo cuenta con caracteres individuales, se utiliza la instrucción *getchar* para leer caracteres y *putc* para escribir caracteres (es de hacer notar que se pueden reedireccionar tanto las entradas como las salidas de un programa, ya que originalmente las entradas son por el teclado y las salidas por el monitor).

Consulta

Es la operación que realiza el usuario para poder leer o acceder a los registros del archivo y conocer su contenido.

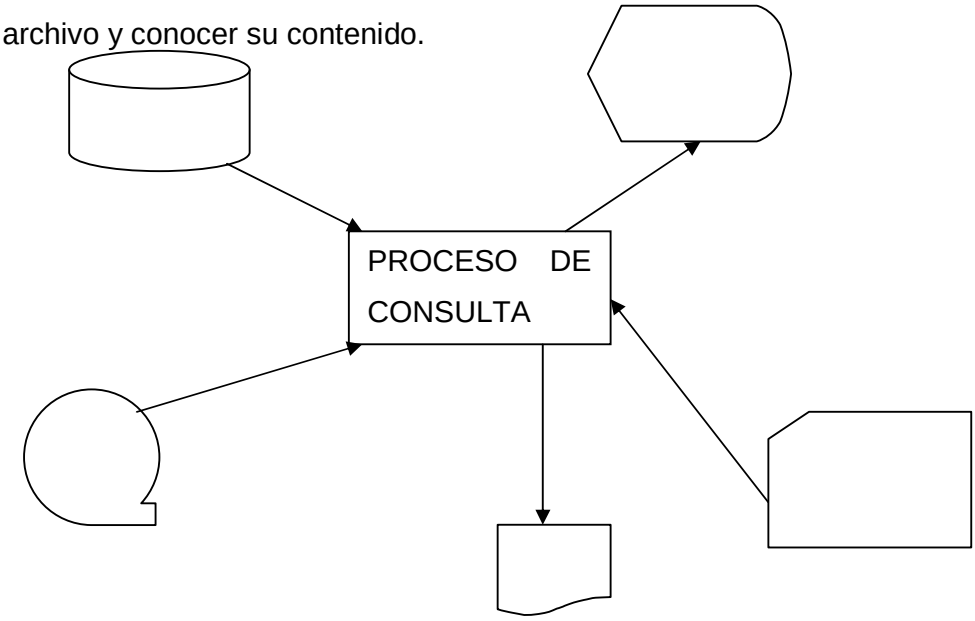


Figura 1.7. Proceso De Consulta

Actualización

La información es poder siempre y cuando ésta sea confiable y *se* encuentre actualizada*,* esto es, que la información se encuentre al día en cuanto a las modificaciones que va sufriendo el archivo.

La actualización de un archivo es la operación que permite precisamente tener al día los registros de un archivo, para lograr esto se requiere realizar las siguientes operaciones en sus registros, que se conocen comúnmente como el **ABC**.

- **A**ltas.
- **B**ajas.
- **C**ambios.

Las **altas** son las acciones de ingresar nuevos registros a los archivos, las **bajas** se dan cuando es necesario desechar algunos registros y los **cambios** cuando se requiere de la modificación de algunos de los datos o campos de un registro.

Por ejemplo, si se tiene un archivo con la información de los usuarios de teléfonos de una ciudad, día con día se va modificando tanto el número de usuarios, como sus datos, ya que puede ser que lleguen nuevos suscriptores, los cuales representaría las **altas**, también puede darse el caso de que algunos clientes ya no desean su servicio telefónico, estos representarían las **bajas**, finalmente, algunos usuarios podrían cambiarse de domicilio dentro de la misma ciudad o simplemente quieran cambiar su número telefónico o cualquier otro dato, siendo clientes de la misma compañía. Este tipo de movimientos que realizan estos clientes representaría los **cambios.**

1.4.2. Clasificación, reorganización

Clasificación

La información se tiene para poderla utilizar y para ello se requiere consultar siendo mas fácil y rápido esta acción, si ésta se encuentra clasificada (ordenada), por alguno(s) de los campo(s) que componen los registros,

pudiendo ser en forma ascendente (de menor a mayor) o descenderte (de mayor a menor).

Este punto, se va a profundizar más en el siguiente tema, donde se abordará con mayor amplitud su importancia.

Reorganización

Esta operación de reorganización consiste en modificar la organización de los archivos, ya que con el paso del tiempo, se puede dar el caso de que se detecten algunos casos que no se contemplaron o que los requerimientos iniciales se modificaron y que se hace necesario cambiar, la organización del archivo, por ejemplo si solo se habían contemplado las actualizaciones en “lote” (*batch*) y se requieren ahora actualizaciones en “línea” y originalmente se tenia un archivo con una organización **Secuencial**, ahora se necesita que se **Directo** o **Secuencial-Indexado.** Para lo cual va a ser necesario realizar una copia del archivo original, para obtener el nuevo ya reorganizado. Solo se refiere al cambio de organización del archivo.

1.4.3. Destrucción

Haciendo la analogía con un ser vivo, un archivo es un ente que se crea, tiene un periodo de vida útil y muere, este último sería la **destrucción,** que es cuando se anula o se borra un archivo, después de esto ya no se puede acceder a ninguno de sus registros.

1.4.4. Fusión

Es la operación que permite juntar varios archivos en uno solo, intercalándose entre ellos, siguiendo algún criterio predeterminado, poniendo por ejemplo el caso de una Universidad, donde cada Facultad pueden tener un archivo propio de alumnos inscritos y se requiere, por parte de Rectoría, tener estadísticas globales de toda esa Universidad, en ese caso conviene fusionar en un solo archivo a los alumnos inscritos para obtener esas estadísticas.

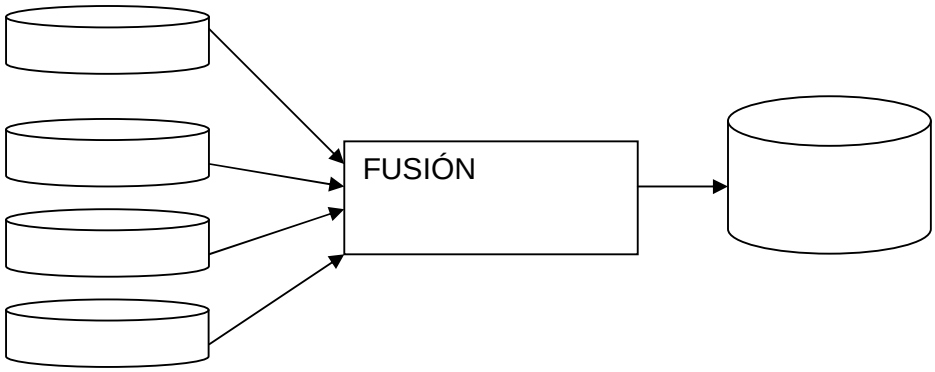


Figura 1.8. Proceso de Fusión

En el siguiente tema se tratará más a detalle este proceso de fusión, ya que existe un método de mezcla, en donde se aplica esta operación.

1.4.5. Rotura/ Estallido

Es la operación contraria a la de fusión, ya que de un archivo se obtienen varios, separándose de acuerdo a algunos criterios.

Retomando el ejemplo anterior (del archivo de alumnos inscritos de una Facultad de cierta Universidad), ,es probable que ahora las estadísticas que se requieran, sean a nivel de las carreras que imparte y entonces, se descompondrá este único archivo en varios, uno por cada carrera que imparte esa Facultad, con lo cual se utilizaría la operación de estallido.

Figura 1.9. Proceso De Rotura/ Estallido

1.5. Mantenimiento de archivos

Se refiere a los cambios que se llevan a cabo en un archivo con el fin de mejorar la ejecución de los programas que lo accedan, este mantenimiento se da por medio de las dos siguientes operaciones:

Reestructuración.

Reorganización.

La **restructuración** consiste en cambiar la estructura del archivo, por ejemplo agregar, quitar o cambiar algún campo de cada uno de los registros de un archivo.

Figura 1.10. Proceso de reestructuración

La **Reorganización** que se trató con anterioridad se puede resumir en el cambio de una reorganización a otra, por ejemplo de la Secuencial a Directa.

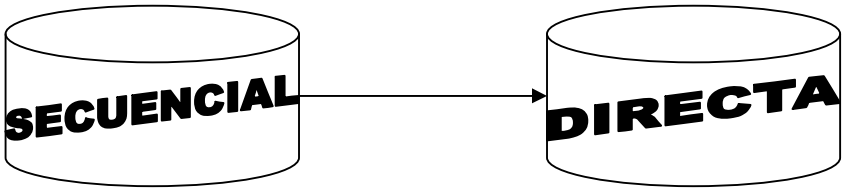


Figura 1.11. Proceso de reorganización

El mantenimiento de los archivos se realiza mediante la ejecución de programas, en donde los archivos se identifican por un nombre externo que está asociado a un nombre interno (lógico) que se utiliza en el programa. Los archivos pueden ser utilizados o compartidos por varios programas.

Para poder utilizar los archivos, los programas tienen que realizar algunas de las siguientes operaciones:

Apertura (abrir) de un archivo. Es donde se indica el dispositivo donde esta o donde se desea que quede almacenado, así como el nombre externo que tendrá físicamente y el nombre interno con que se manipulará dentro del programa, también el tipo de organización que tendrá dicho archivo. Algunos Sistemas de Archivos permiten definir los parámetros vistos anteriormente como el MAXRECISIZE, BLOCKSIZE, etc.

Lectura (*read o scanff*) de un registro. Va a depender del tipo de organización del archivo (Secuencial, Directa o Secuencial-Indexada), como lo realice.

Escritura (*write o printf*) de un registro en el archivo, el cual queda después del último registro del archivo.

Reescritura (*rewrite*). Consiste en reescribir en la misma posición el mismo registro, tal vez modificado en algún o algunos de sus campos. No todos los tipos de organización lo permiten.

Rebobinado (*rewind).* Es la acción de regresar al inicio del archivo, no importando en que posición se encuentre la lectura o escritura actual.

Cierre. Es la operación de guardar el archivo, ya que si se deja abierto, puede llegarse a perder parte o toda la información que contiene.

Borrado (*delete*) de un registro. No todas las organizaciones de archivos lo permiten.

Destruir un archivo. Se da cuando el archivo con sus características o su información dejan de ser útiles y tienen que ser destruidos.

1.6. Algoritmos para manipular archivos secuenciales

En los archivos secuenciales los registros, se van agregando uno detrás de otro, esto es en orden cronológico de llegada, uno a continuación de otro y terminan igual que cualquier organización con una marca de fin de archivo **EOF** (por sus siglas en inglés *end of file*). Al agregar registros, se harán después del último que tenga el archivo.

Es importante recordar que este tipo de archivos ocupan menos memoria, son muy empleados para el almacenamiento de información, donde los contenidos sufren pocas modificaciones en el transcurso de su vida útil y para procesos de actualización “en lote”.

Las operaciones básicas que permite un archivo secuencial son:

- Creación
- Consulta
- Actualizaciones (altas, bajas y cambios)

Creación

Es un proceso secuencial, en el que los registros se van almacenado en el orden en que van llegando al archivo. El algoritmo es muy sencillo, sólo se requiere de un algoritmo implantado en un programa, donde se leerán los datos de cada registro y posteriormente se grabarán en un archivo.

Puede darse el caso de que se cree el archivo nuevo, desde el primer registro, o ir agregando registros a continuación del último registro a un archivo que ya exista.

Para crear un archivo se requiere de las siguientes acciones:

- Abrir un archivo.
- Leer datos del registro.
- Grabar el registro en el archivo.
- Cerrar el archivo.

Nótese que cuando se abre un archivo a través de los programas de aplicación, se ocupan algunos atributos donde se especifica, entre otras, las siguientes características:

- Si existe el archivo que borre su contenido, en caso contrario lo cree nuevo.
- Si existe el archivo, que conserve sus datos y empiece a grabar después del último.
- Si el archivo va a ser ocupado de lectura o de escritura.

Estos atributos son los que se van a utilizar en los algoritmos que a continuación se presentan en pseudocódigo.

Algoritmo de Creación

```
inicio
        desplegar " 1.- Creación de un archivo nuevo"
```

```
        desplegar " 2.- Añadir datos al archivo"
        leer opción
        si opción es igual a 1 entonces
                abrir el archivo para crearlo nuevo
        en caso contrario
                abrir el archivo para añadir datos
        fin
        mientras no sea fin de archivo hacer
                leer los datos de un registro
                escribir esos datos en el archivo
        fin
        cerrar el archivo
fin
```

Consulta

Esta operación va muy relacionada con lo que son las búsquedas, las cuales pueden ser por comparación o por transformación de llaves.

En el caso de los archivos Secuenciales, sólo se podrán hacer por comparación de llaves que consiste en la recuperación de los datos de un registro específico a consultar dentro del archivo. El tema de búsquedas, se va a tratar con más detalle en el tercer tema de este trabajo.

En la consulta en **archivos Secuenciales**, se aplicará el método de comparaciones de llaves que consiste en leer desde el primer registro del archivo hasta compararlo con el que se está consultando, si es igual, entonces termina la búsqueda o consulta con éxito y los datos son los del registro encontrado, en caso contrario, se lee el siguiente registro y se vuelve comparar, así sucesivamente hasta que se encuentre el registro buscado o se llegue al fin del archivo (**EOF),** si esto sucede, indicará que no se encontró el registro dentro del archivo.

Es importante resaltar que la búsqueda o consulta se puede realizar por uno o varios campos que conforman el registro del archivo.

Las consultas por comparación de llaves pueden ser muy rápidas si el registro se encuentra en las primeras posiciones del archivo o, por el contrario, muy lentas si se encuentra en las últimas posiciones y si además el archivo es muy grande (cientos de registros). El peor de los casos es cuando no se encuentra el registro dentro del archivo, sin embargo, para percatarse de ello es necesario recorrerlo todo, consumiendo tiempo en la búsqueda.

A continuación se muestran dos algoritmos, el primero de ellos consulta todo el archivo y lo despliega; el segundo sólo consulta y despliega un registro especifico que se busca en el archivo.

Algoritmo de consulta a todo el archivo

```
inicio
        abrir archivo para lectura
        leer un registro
        inicializar N = 1
        mientras no se encuentre el fin de archivo (EOF)
                desplegar el registro
                leer registro del archivo
                N=N+1
        fin
        desplegar “el número de registros del archivo es de “ N -1
        cerrar el archivo
fin
```

Algoritmo de consulta a un determinado registro con una clave x

```
inicio
        abrir archivo para lectura
        leer un registro
```

mientras no se encuentre el fin de archivo (**EOF**)
si registro (campo x) = registro (campo leído) entonces
desplegar “búsqueda con éxito ya que el registro buscado existe”
cerrar el archivo
alto
fin
leer registro del archivo
fin
desplegar “búsqueda sin éxito, no existe el registro”
cerrar el archivo
fin

Actualizaciones (altas, bajas y cambios)

Las actualizaciones requieren de las operaciones del A, B, C (altas, bajas y cambios).

Altas

Una **alta** es la creación de un nuevo registro en el archivo, el cual se trató en la operación de creación. En el caso particular de un archivo Secuencial, la operación de **alta**, es la creación de un nuevo archivo, ya que este tipo de organización, no admite la incorporación de nuevos registros intermedios, sólo después del último.

A continuación se muestra el algoritmo de altas, que es una modificación del de creación:

Algoritmo de altas
inicio
abrir archivo para añadir
leer un registro
mientras no se encuentre el fin de archivo (**EOF**)

escribir registro en el archivo
leer un registro
fin
cerrar el archivo
fin

Bajas

Es la supresión de un registro del archivo, pero con este tipo de organización no se puede dar la baja directamente, por lo que será necesario realizarla con la utilización de un archivo auxiliar, el cual también es secuencial.

Para dar la baja es necesario utilizar un archivo secuencial auxiliar, en el cual se copiará el archivo original, sin grabar en él los registros que se deseen dar de baja, no se dan de baja directamente, únicamente se dejan de escribir en el archivo auxiliar. Al final se tendrán dos archivos: el **original** y el **auxiliar,** que ya no tendrá los registros que se dieron de baja.

Algoritmo de bajas
inicio
abrir archivo_ auxiliar para crearlo nuevo
abrir archivo_ original para lectura
leer un registro_ original
mientras no se encuentre el fin de archivo (**EOF**)
si registro_ original (campo_ tipo_ mov ≠ “baja”) entonces
escribir registro_ original en el archivo_ auxiliar
fin
leer registro_ original
fin
cerrar el archivo_ auxiliar
cerrar el archivo_ original
fin

Figura 1.12. Proceso de baja de registros en archivos secuenciales

Este procedimiento de bajas también se puede llevar a cabo por medio de un arreglo (array) de una dimensión, donde se almacenarán todos los registros del archivo original, donde se marcará aquel o aquellos registros que se deseen dar de baja y posteriormente se irán grabando cada uno de los elementos del arreglo no marcados como baja al nuevo archivo auxiliar.

Cambios

Este proceso consiste en la localización de un registro para realizar algún cambio en uno o varios de los campos y, posteriormente, volver a reescribir (*rewrite*) el nuevo registro en el archivo.

Pero como en un archivo secuencial, no se puede dar esta operación directamente, se necesitará nuevamente la utilización de un archivo auxiliar, donde se escribirán tanto los registros que no lleven ninguna modificación, como los que ya fueron modificados.

A continuación se muestra el algoritmo en pseudocódigo, que lo realiza.

```
Algoritmo de Modificación
inicio
        abrir  archivo_ auxiliar para crearlo nuevo
        abrir archivo_ original para lectura
        leer un registro_ original
```

```
        mientras no se encuentre el fin de archivo (EOF)
                escribir "Modificar S/N"
                leer la respuesta
                si respuesta = 'S'  entonces
                        realizar la modificación del o de los campo(s)
                fin
                escribir registro_modificado en el archivo_auxiliar
        fin
        cerrar el archivo_auxiliar
        cerrar el archivo_original
        borrar archivo original
        cambiar el nombre del archivo auxiliar por el del original
fin
```

1.7. Algoritmos para manipular archivos Indexados

En esta organización es posible el acceso a un registro en particular (aleatoria) y al proceso secuencial desde el inicio del archivo o de cualquier otro registro del mismo.

Los registros que se encuentran en ese este tipo de archivos se identifican ya sea por medio de un número único o por un conjunto de caracteres exclusivos, lo cual se conoce como clave o llave primaria.

Es importante resaltar que un archivo Indexado puede manejar una llave primaria, pero también puede tener más de una llave de acceso, conocidas como llaves secundarias o alternas.

En este tipo de archivos se puede realizar el procesamiento en forma aleatoria, es decir, los registros pueden ser almacenados en cualquier orden, no precisamente deben de coincidir con su distribución física, pero también se pueden almacenar de acuerdo a una secuencia física, la cual puede ser de acuerdo al valor de la clave o llave primaria.

Un **archivo Indexado**, es un conjunto de **n** registros, donde cada uno de ellos cuenta con una llave que identifica de manera única a cada registro y una información que se asocia a esta llave. Por ejemplo, se tiene el número de cuenta de un alumno, es la llave y la información asociada a ella puede ser el nombre del alumno, número de créditos, clave de la carrera, etc.

Generalmente, la llave tiene una longitud más corta que el resto de la información asociada, y el conjunto de todas las llaves se tiene en la memoria principal.

Para acceder los registros en forma aleatoria, tanto para lectura como para escritura, se hace necesaria la creación de un índice, para su recuperación.

El índice es una entidad que tiene como entrada la llave primaria cuya salida es una información que permite la rápida localización física del registro. Los índices son almacenados en un archivo de índices, con el fin de optimizar las búsquedas.

Este archivo de índices, se asocia al resto de los datos a través de un apuntador.

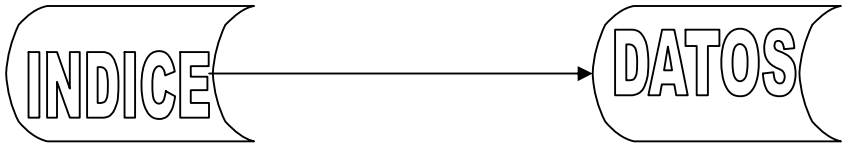


Figura 1.13. Archivo de índices y de datos.

Para manipular los archivos Indexados, se utiliza la estructura de índices como un árbol de búsqueda binaria para los índices. En la siguiente figura, se ejemplifica este concepto.

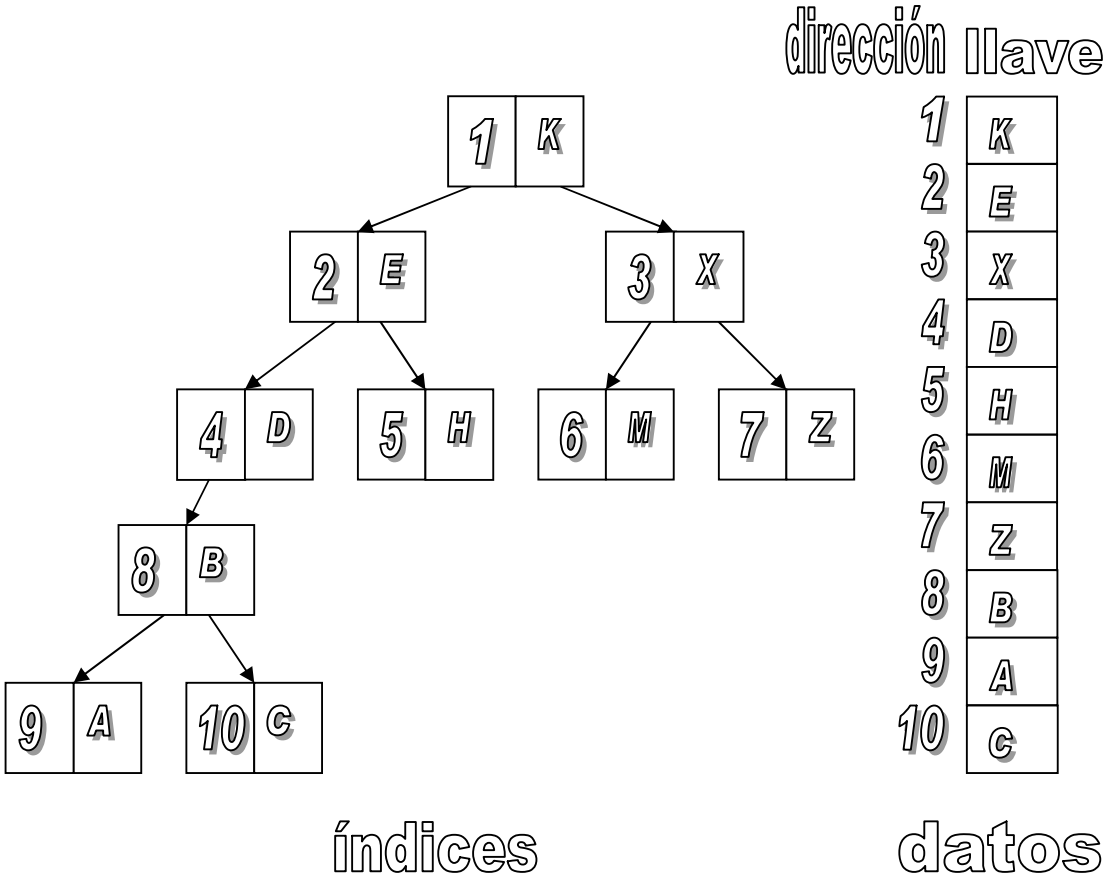


Figura 1.14. Árbol de búsqueda.

La búsqueda se realiza a través del árbol binario de búsqueda. Este tipo de árbol y sus búsquedas se verán detalladamente en el tema 3.

El acceso secuencial a este tipo de archivos se puede llevar a cabo a través de un recorrido **inorden** del árbol binario de búsqueda.

Otra técnica para poder manipular los archivos secuenciales indexados, es a través de árboles B y B+ los cuales van a tener una raíz, nodos intermedios y hojas, que son aquellos nodos que no tienen ningún descendiente.

Utilización de árboles B+: en esta otra forma, los datos (o llaves) sólo se van a encontrar en los nodos hojas (también conocidos como nodos terminales) y los nodos, que son la raíz y los intermedios, van a servir para acceder a los nodos terminales, esta técnica se muestra en la siguiente figura:

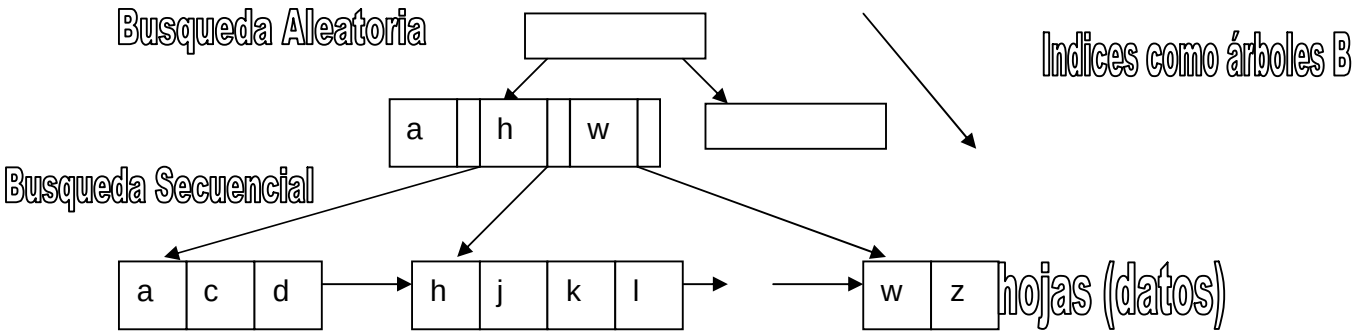


Figura 1.15. Árbol de búsqueda B+.

Los nodos índices y los nodos hojas pueden llegar a tener diferentes formatos y/o tamaños. Los **nodos hoja son** ligados de izquierda a derecha, es en estos nodos donde se encuentra la llave, la lista de nodos hojas se procesa secuencialmente y la búsqueda se realiza de la raíz del árbol a través del índice, hasta la hoja que lo encuentre.

En la siguiente figura se muestra detalladamente un archivo secuencial indexado, con el esquema de árboles B+, donde los índices y los datos son organizados en bloques. Los índices están organizados en un árbol y los datos tienen una estructura secuencial.

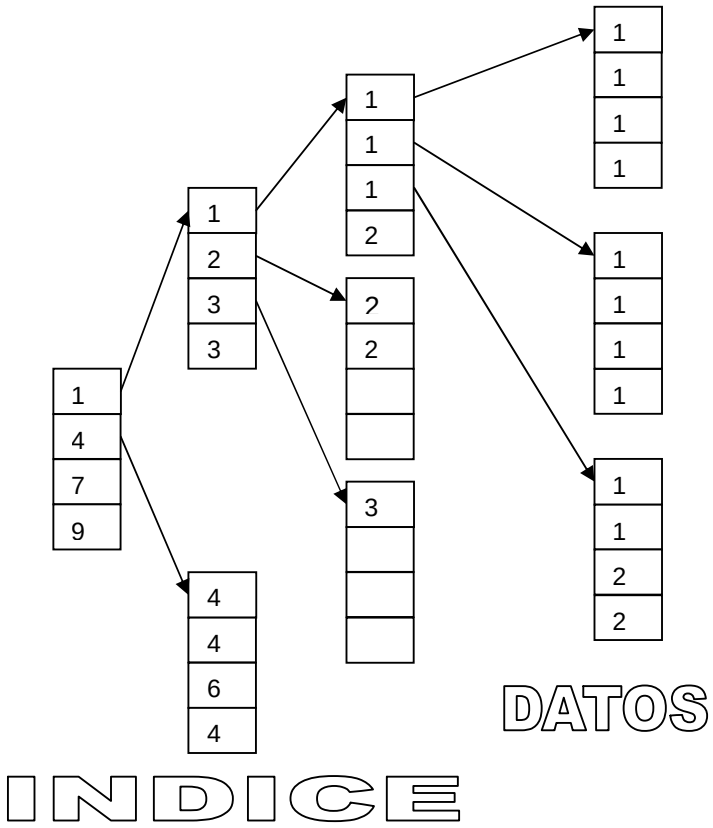


Figura 1.16. Árbol de búsqueda B+ con varios niveles.

1.8. Tratamientos de colisiones

Antes de tocar las colisiones es conveniente definir a grandes rasgos lo que son las **funciones de dispersión o de h*ash***, las cuales tienen como finalidad transformar, a partir de operaciones numéricas, los caracteres que componen la llave (o clave) del registro, de tal forma que indiquen en qué dirección se va a almacenar o recuperar dicho registro.

Estas direcciones idealmente deben ser únicas para cada valor de llave e irrepetibles. Por esto, también se conocen como **funciones de mapeo** de uno a uno (un valor de llave para una dirección única dentro del archivo).

El problema es que no siempre se puede dar esta relación de uno a uno, pero no hay de qué preocuparse, tiene solución.

Cuando se tiene más de una llave diferente y se transforman a través de la función de dispersión y de la misma dirección para almacenar estos registros se conoce como **colisión.** Ya que sólo se puede almacenar un registro en cada dirección.

La solución de colisiones se pude dar de dos maneras:
1. Direccionamiento Abierto.
2. Encadenamiento.

Se profundizará más sobre colisiones y su tratamiento en el tema 3.

1.9. Acceso a archivos directos mediante indexación

La lectura y/o escritura en este tipo de organización es rápida, ya que se accede directamente al registro deseado y no se necesita recorrer los registros anteriores, como sucede en los secuenciales.

Las operaciones con archivos de acceso directo son las siguientes:

Creación

Es muy parecida a las organizaciones Secuenciales y Directas, el proceso de creación es la acción de ir introduciendo los registros nuevos en el dispositivo, que los va a almacenar y en la dirección previamente obtenida, después de ocupar el algoritmo o función de *Hash* (que tiene como entrada la llave del registro y da como salida la dirección, donde se va a almacenar o recuperar dicho registro).
Si la dirección de ese registro ya está previamente ocupada por otro registro, puede resolverse de diferentes maneras, de acuerdo al esquema que se utilice (por ejemplo Mapeo por Direccionamiento absoluto o Mapeo o por Direccionamiento relativo, por Directorio o por cálculo) o bien si se maneja un espacio de excedentes, el nuevo registro quedará ahí.

Algoritmo de creación

inicio

abrir el archivo para escribir

leer un registro

mientras no sea fin de archivo (o de introducción de datos) hacer

calcular la dirección, mediante algoritmo de conversión

verificar si la dirección esta libre

¿si esta libre la dirección?

se graba el registro

en caso contrario

buscar una dirección en el área de excedentes

grabar el registro

fin

leer un registro

fin

fin

Algoritmo de altas

Para este algoritmo, se necesita introducir el contenido y la posición en la cual se desea grabar el registro **NR** (Número del Registro o posición donde se desea grabar)

inicio

abrir el archivo para escritura

repetir

leer 'número de la posición, donde se desea dar de alta el registro' NR

si (NR $\geq$ 0) OR (NR $\leq$ LimiteDeRegistrosDelArch) entonces

leer registro NR

Si Bandera = 1 entonces

escribir 'registro ya existe y no se graba'

en caso contrario

Bandera =1

leer datos del registro

escribir Bandera y datos en registro NR

fin

en caso contrario

escribir ‘error el **NR** (número de registro) fuera de rango del arch.’

fin de si

hasta que no se deseen más altas

cerrar el archivo

fin

Algoritmo de bajas

Para realizar las bajas, no se borra físicamente el registro, lo que se hace es marcar el registro en un campo, que para el siguiente algoritmo se usará como **bandera**, que tendrá un valor de **cero** cuando no exista y valor de **uno** cuando no exista lógicamente

inicio

abrir archivo para escribir

repetir

leer NR (número de registro)

si (NR $\geq$ 0) OR (NR $\leq$ LimiteDeRegistrosDelArch) entonces

leer registro NR

si Bandera = 0 entonces

escribir ‘el registro no existe, no se da de baja’

en caso contrario

Bandera = 0

Escribir el registro NR (ya modificada la Bandera)

fin

en caso contrario

escribir ‘registro fuera del rango del archivo’

fin

hasta que no se deseen más bajas

```
        cerrar el archivo
fin
```

Algoritmo de cambios

En este tipo de archivos, para realizar los cambios o modificaciones, primero se lee el registro, posteriormente se modifica y finalmente se reescribe *(rewrite*).

```
inicio
        abrir archivo para reescribir (de lectura y escritura)
        repetir
                leer NR (número de registro)
                si (NR ≥ 0) OR (NR ≤ LimiteDeRegistrosDelArch) entonces
                        leer registro NR
                        leer los cambios de lo(s) campo(s)
                        realizar los cambios
                        reescribir el registro en el archivo con los cambios
                en caso contrario
                        escribir 'registro fuera del rango del archivo'
        hasta que no se deseen más modificaciones
        cerrar el archivo
fin
```

Resumiendo los conceptos vertidos en este primer tema, se puede decir, que los archivos surgieron por la necesidad de poder almacenar por tiempo prácticamente indefinido (no sólo mientras durase la ejecución del programa que lo generase) y para tener una mayor capacidad de almacenamiento de datos y de información, estos requerimientos, siguen estando presentes en lo que a desarrollo de sistemas de información se refiere.

Han aparecido también nuevas Tecnologías de la Información (TI), como los son los manejadores de Bases de Datos que en un principio se pensaba que llegarían a desplazar el uso de los archivos, sin embargo, estos Manejadores se auxilian del Sistema de Archivos y es evidente que los Manejadores de Bases de Datos tienen un costo extra, ya que hay que comprar un software adicional, en cambio, utilizar el Sistema de Archivos no tiene costos adicionales, basta con tener el sistema operativo.

Cualquier aplicación de sistemas de información se puede realizar si se cuenta con un Sistema Operativo y un Compilador (o en su caso Intérprete) de algún lenguaje de programación, siendo a través de él que se pueden manipular los archivos. Se invierte más tiempo y esfuerzo en desarrollar las aplicaciones con sólo la utilización del Sistema de Archivos a que si fuera con un Manejador de Base de Datos, pero se pueden llevar a buen término.

En tanto que si se cuenta con un Manejador de Base de Datos, esas aplicaciones se realizarán más rápidamente, pero se dependerá mucho de éste.

Bibliografía del tema 1

ALCADE Lancharro, Eduardo y GARCÍA López, Miguel, *Métodología de la programación*, España, Mc Graw-Hill, 1992, pp. 416.

BRAVO DE CAIRO, Carmen y STEINBERG NOSNIK, Isarel, *Tecninotas. Cómo hacer eficientes los archivos en la B-6700, B-6800 y B-7800*, México, Centro de Servicios de Cómputo CSC UNAM, Febrero de 1981, pp. 11.

EUÁN Ávila, Jorge Ivan y CORDERO Borboa, Luis Gonzaga., *Estructuras de datos*, México, Limusa, tomada de la primera edición de la UNAM (Por la unidad Editorial de la FACULTAD DE INGENIERÍA), 1989, pp. 219.

REYES González, Armando, *Tecninotas. Atributos de archivos*, México, Centro de Servicios de Cómputo CSC UNAM, Febrero de 1982, pp.11.

ROSE Gómez, César E., *Archivos, organización y procedimientos,* México, Computec**,** 1993, pp. 227.

VILLERS, Abelardo**,** *Tecninotas. Introducción a Base de Datos*, México, Centro de Servicios de Cómputo CSC UNAM, Septiembre de 1981, pp. 11.

Actividades de aprendizaje

A.1.1. Elabora un mapa conceptual de este tema.

A.1.2. Investiga Bibliografía actual, diferente a la mencionada en este capítulo, indicando la Ficha Bibliográfica como se mostró anteriormente y si es posible, la clasificación hecha por la biblioteca y en cuál de ellas, se localizó, así como los conceptos o palabras claves que creas más conveniente incluir por su relevancia, con los temas de la materia, para su posterior consulta.

A.1.3. Realiza un programa en el lenguaje de programación **C** para implementar uno de los algoritmos que permita manipular archivos secuenciales, a partir del pseudocódigo visto en esa sección.

A.1.4. Elabora un programa en el lenguaje de programación **C** para implementar uno de los algoritmos que permita manejar archivos directos, tomando como base el pseudocódigo visto en este tema.

A.1.5. Investiga por lo menos una aplicación donde se ocupen archivos secuenciales.

Cuestionario de auto evaluación

1. ¿Qué es un archivo?
2. ¿Qué es el Factor de Bloqueo (FB)?
3. ¿Qué es un tipo de datos?
4. De acuerdo a su forma de acceso ¿Cómo se clasifican los archivos?
5. ¿Cuáles son las dos operaciones en el mantenimiento de archivos?
6. ¿Cómo se define la organización de los archivos?
7. Menciona a qué se enfocan los tipos de datos abstractos.
8. ¿Cuál es la diferencia entre el Direccionamiento absoluto y el Direccionamiento relativo en la organización Directa de archivos?

9. ¿Qué campos de información contiene el directorio o diccionario en los Archivos Directos?
10. ¿De qué depende la ejecución de una función *hash*?

Examen de autoevaluación

Relaciona la columna de la izquierda con la de la derecha y coloca la respuesta que consideres correcta dentro del paréntesis.

( ) 1. Los datos se obtienen por:	a. Información.
( ) 2. Un registro es:	b. Los algoritmos de PUSH y POP.
( ) 3. Los datos transformados se convierten en:	c. Los dispositivos de entrada
( ) 4. La llave o clave es:	d. El número de caracteres, palabras o campos, que se obtienen al ejecutar una lectura o escritura.
( ) 5. Registro lógico es:	e. Un campo de identificación único del registro.
( ) 6. El Factor de Bloqueo se da por:	f. Tipos de datos abstractos.
( ) 7. TDA es la abreviación de:	g. Sólo se leen o se escriben, pero nunca las dos operaciones al mismo tiempo.
( ) 8. Son un ejemplo de TDA	h. Al aplicar la función de *hash*, da la misma dirección, para más de una llave diferente.
( ) 9. Una característica de los Archivos secuenciales es:	i. Un conjunto finito de campos.

( ) 10. Una colisión se da cuando:	j. La relación del tamaño del registro físico entre el tamaño del registro lógico.

Examen de autoevaluación (segundo apartado)

1. Son algunas de las etapas de la administración de datos:
   a) captura, validación y clasificación.
   b) búsquedas, comparaciones y distribuciones
   c) consulta, alta y baja.
   d) cambios, inserciones y supresiones.

2. Es una de las características de los archivos:
a) residencia en memoria principal y memoria cache.
b) su permanencia sólo dura el tiempo de ejecución del programa.
c) sólo puede ser utilizado por un sólo programa de aplicación.
d) gran capacidad de almacenamiento, casi ilimitado.

3. Entre las ventajas del uso de archivos se tiene:
a) permitir un sólo acceso a la vez a sus registros.
b) mayor portabilidad de los datos de una computadora a otra.
c) el uso de un lenguaje estructurado de consulta (sql).
d) el uso del protocolo FTP.

4. Entre las desventajas en el uso de archivos se encuentra:
a) el control de candados para el acceso a memoria principal.
b) la continúa aparición de colisiones en el acceso secuencial.
c) mayor inversión de trabajo para aplicaciones con archivos.
d) la mayor cantidad de memoria para las estructuras de datos.

5. Un campo es:

a) un grupo de caracteres.

b) una serie de áreas.

c) un conjunto de registros.

d) una colección de factores de bloqueo.

TEMA 2. MÉTODOS DE CLASIFICACIÓN Y CONSIDERACIONES DE COMPLEJIDAD

Objetivo particular.

Al término de este capítulo el alumno será capaz de:

Identificar lo que es el proceso de clasificación, también conocerá y aplicará los métodos internos y externos de clasificación (ordenamientos), así como algunos criterios, para seleccionar uno de ellos.

Temario detallado

2.1. Ordenamiento por intercambio (*Bubblesort*)
2.2. Ordenamiento por inserción directa
2.3. Ordenamiento por selección
2.4. Método Shell
2.5. Ordenación rápida (*Quick Sort*)
2.6. Criterios de selección del método de ordenamiento
2.7. Análisis comparativo de las complejidades de los distintos métodos de ordenamiento

Introducción

La importancia de contar con información oportuna y a un costo accesible, radica en poder ayudar en la toma de decisiones dentro de una empresa, para lo cual es necesario transformar, a través de procesos, los datos (provenientes de diversos eventos) en información, para lo cual juega un papel muy importante el tener la información clasificada.

Algunos ejemplos donde la clasificación de la información es de vital importancia son los que a continuación se mencionan:

- En el uso de cualquier diccionario escolar, donde sería casi imposible localizar una palabra que no tuviera una clasificación alfabética.
- En la búsqueda de algún libro, revista o tesis en una biblioteca, ya que con la gran cantidad de ejemplares que manejan, sin alguna clasificación, sería casi imposible la localización del ejemplar que se estuviera buscando.

- En la generación de documentación como estados de cuenta bancarios, listas de los alumnos inscritos, inventarios etc., donde resulta de gran ayuda, el tener la información clasificada de acuerdo a alguna(s) llave(s).
- En la actualización de la información, a través de las de **altas, bajas y cambios,** que se hacen sobre sus registros**,** las cuales se pueden realizar con mayor eficiencia, si la información esta clasificada.
- En la mezcla de dos o más archivos secuenciales, en uno solo Fusión, la cual fue comentada en el primer tema, se realiza más rápidamente si los archivos también estuvieran previamente clasificados.

Así se podrían seguir mencionando una gran cantidad de ejemplos, pero se puede resumir que la clasificación de la información, permite realizar búsquedas y actualizaciones de la información de una manera más sencilla y rápida, por lo que el tema de clasificación adquiere una gran importancia e interés.

Clasificación (ordenamientos)

El proceso de clasificación "consiste en ordenar una secuencia de registros de tal forma que los valores de sus claves formen una secuencia no decreciente. Esto es, dados los registros $r_1, r_2, \ldots r_n$, con valores de clave $k_1, k_2, \ldots k_n$, respectivamente , debe resultar la misma secuencia de registros en orden $r_{i_1}, r_{i_2}, \ldots r_{i_n}$, tal que $k_{i_1} \leq k_{i_2} \leq \ldots \leq k_{i_n}$. No es necesario que todos los registros tengan valores distintos, ni que los registros con claves iguales aparezcan en un orden particular"[7].

También se puede definir como

"ordenar una estructura de datos es establecer un orden de precedencia entre los elementos de la estructura, de acuerdo a uno o más campos (llaves) que se

[7] Alfred V Aho, et al, Estructuras de Datos y algoritmos, p. 253.

seleccionen para tal fin. Para lograr esto, se han desarrollado una gran cantidad de algoritmos".[8]

En primera instancia, parecería que el ordenar, buscar información o datos es algo sencillo, trivial, que no tiene gran complejidad, pero puede a llegar a ser una operación costosa, tanto en dinero como en tiempo, por lo que la búsqueda y/o la clasificación se harán cuando sea estrictamente necesarios.

Es evidente que a veces se ocupan indistintamente los términos ordenar y clasificar, dependiendo del autor o de la traducción al español, pero hay autores como Donald Knuth que prefieren utilizar sólo el término de clasificación. Para evitar confusiones por lo cual se utilizará en el resto de estos apuntes sólo la palabra **clasificación**.

Los métodos de clasificación, se pueden dividir en dos grupos:
Los **internos**.- Sólo ocupan memoria principal.
Los **externos**.- Ocupan memoria principal y memoria secundaria.

Los internos se utilizan cuando el volumen de información o datos, es relativamente pequeño, de tal forma que caben todos en la memoria principal.

Los externos se ocupan cuando se tiene tal cantidad de datos o de información que no caben en memoria principal y es necesario auxiliarse de la memoria secundaria y, en algunos casos, de los métodos internos.
Algunos autores hacen una separación de los métodos de clasificación, donde aparecen los nombres de los grupos y de los métodos, como la que a continuación, se muestra[9]:

---

[8] Jorge Iván Ávila Euán y Luis Gonzaga Cordero Borboa, *Estructuras de datos*, p. 40.

[9] Idem.

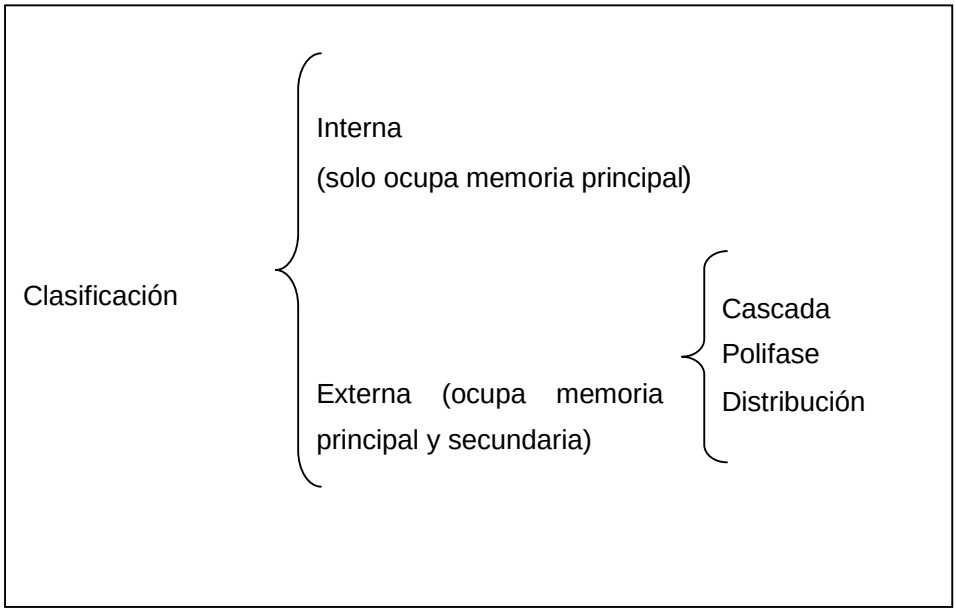


Figura 2.1. Grupos de clasificación.

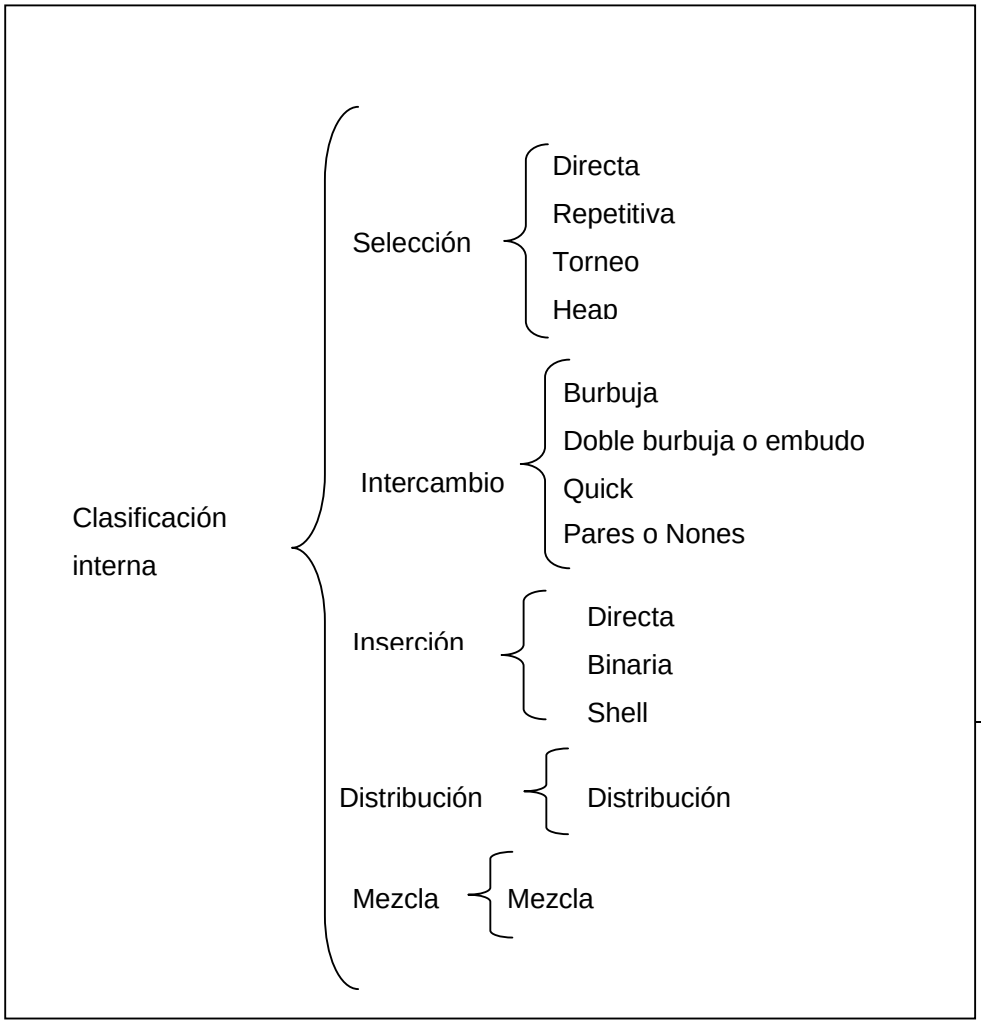


Figura 2.2. Tipos de clasificación interna.

Es importante considerar que la mayoría de las aplicaciones donde se utiliza la clasificación, la información que tienen los registros o nodos, puede ser grande (por ejemplo todos los datos de un cliente de tarjetas de crédito de un banco), y sólo se ocupa la llave principal y/o alguno(s) otro(s) campos, para realizar la clasificación, como se verá en los ejemplos que se mostrarán más adelante, donde se considerará sólo la llave, para que sea más rápida su explicación.
Es de hacer notar que los métodos que se describen a continuación, pueden ser utilizados, tanto para clasificar en forma ascendente como decentemente, solo habrá que cambiar algunas líneas de código, donde se realizan las comparaciones.

En cada algoritmo que se describa se indicará qué criterio se seguirá: ascendente o descendente.

A continuación se describirán algunos de los métodos que se mencionaron anteriormente, así como su codificación en el lenguaje de programación ***C***; para poder realizar con criterios uniformes los métodos que se analizarán se utilizarán los mismos datos, los cuales se generarán aleatoriamente y se guardarán en un archivo a través del primer programa que se presenta.

Este archivo con 15 números (llaves) desordenados será utilizado en todos programas de clasificación, donde venga el código en **C**.

Para poder optimizar el espacio de este trabajo, sólo se presentarán las partes medulares de cada programa.

Se utilizará un archivo con números enteros, generados aleatoriamente con el programa que utilizará la función **rand** para generarlos, los cuales estarán entre cero y noventa y nueve y los guardará en un archivo secuencial que tiene el

nombre **aletorios.txt**. El código del programa que genera este archivo se muestra a continuación.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main()
{
   int k, a1[10];
   time_t t;
   FILE *ale;
   ale=fopen("aleatorios.txt","w");
   if(ale==NULL)
   {
     printf("****ERROR al abrir  el archivo  aleatorios.txt****");
     exit(EXIT_SUCCESS);}
   srand((unsigned) time(&t)); // Se generan los 50 numeros aleatorios
   for (k=0;k<50;k++)
      fprintf(ale,"%d \n",rand() % 100);
   fclose(ale);
```

También es conveniente recalcar, que se estarán usando arreglos, para ejemplificar los métodos, partiendo que se pasarán estos números aleatorios a un arreglo de una dimensión y que se utilizarán en el lenguaje de programación **C,** o alguno que parta desde el índice **cero** en el arreglo que manipulen.

2.1. Ordenamiento por intercambio (B*ubblesort*)

Definición del método

El método ***Bubblesort*** es uno de los más conocidos y populares, por la facilidad de su algoritmo y programación, así como su rapidez de ejecución cuando se trata de pocos datos.

Este algoritmo se basa en comparar las llaves que se encuentran en las posiciones n y n-1, siendo n la última posición y realizando un intercambio (en caso de clasificar en forma **ascendente**) si *n es < que (n-1)* y después compara las posiciones (*n-1) y(n-2)*, intercambiándolas si es que *(n-1) es <*

que (n-2) y así sucesivamente, hasta que se comparan los datos que se encuentran en las posiciones uno y dos, siendo cuando termina la primera pasada* y es cuando se coloca un elemento en la primera posición, el cual es marcado de alguna manera, para que en las siguientes pasadas, ya no se tome en cuenta.

Para la segunda pasada, se repite el mismo proceso en la lista de elementos restantes y se coloca el elemento más pequeño en la segunda posición.

En cada pasada se va colocando un elemento, a excepción de la última, en la cual se colocan dos elementos ya que al tener sólo dos, al colocar uno de ellos en su posición de acuerdo a su valor, el otro automáticamente queda en la posición que le corresponde.

El algoritmo es muy sencillo de programar, no ocupa espacio adicional, pero tiene la característica de que cuando el número de elementos crece, también aumenta el tiempo de su ejecución y es muy rápido cuando se trata de pocos elementos los que se van a clasificar.
El nombre de **B*ubblesort*** (Burbuja), se debe a que “emerge” de la posición en la que se encuentre el elemento más pequeño hacia la primera de la lista, si se clasifica en forma **ascendente**, o “emerge” el elemento más grande, si se clasifica en forma **descendente**.

El código del algoritmo es el siguiente:

```
/*  Funcion que utiliza el metodo de la BURBUJA    */
void burbuja(int a[], int num, int z)
{
   printf("\f \n\t\t");
   int c=0,i=0,j,k, bandera, aux;
```

* Una pasada, es el proceso que realiza el algoritmo, para poder colocar uno de los elementos a clasificar en la posición que le corresponde, de acuerdo al criterio que se este siguiendo, ya se en forma ascendente o descendente.

```
   printf("\n\t**** METODO DE LA BURBUJA *****\n\n");
   printf("\n\t*** los elementos desordenados son ***\n");
   for (k=0;k < num; k++)
          printf (" %d ",a[k]);
   printf("\n\n");
   principio = clock();
   i=1;
   aux=0;/*variable auxiliar para el intercambio de valores */
   k=-1;
   bandera=1; // bandera que se prende  para terminar el ciclo
   while (bandera)
   {
      k=k++;  // k indica la posicion del elemento que se va colocando
      bandera=0; //se hace falso para terminar el ciclo de clasifi.
      for (j=num-1;j > k;j--)
      {
       delay(10);
       if (a[j] < a[j-1])
       {
          bandera=1;
          aux=a[j];  /* realiza el intercambio de elementos*/
          a[j]=a[j-1];
          a[j-1]=aux;
       }
      }
    }
   fin = clock();
   num=num-1;
    printf("\n\t*** los elementos ordenados en forma");
    printf(" asecendente son ***\n");
    for (k=0;k <= num; k++)
          printf (" %d ",a[k]);
   nele[1][z]=num+1;
   ntiempo[1][z]=((fin - principio)) / CLK_TCK ;
   printf("\n\n");
   printf("\n\t*** El tiempo de ejecucion en decimas de segundos\n");
   printf("\t\ten forma ascendente fue de: %7.5f ",((fin - principio))
/ CLK_TCK);
   printf("****");
```

```
}
```

La salida es la siguiente:

```
        **** METODO DE LA BURBUJA *****
        *** los elementos desordenados son ***
 62   39   62   77   77   86   74   55   82   86   3   88   44   40   9
        *** los elementos ordenados en forma ascendente son ***
 3   9   39   40   44   55   62   62   74   77   77   82   86   86   88

        *** El tiempo de ejecucion en decimas de segundos
                en forma ascendente fue de: 0.82418 ****
```

Ejemplo

A continuación se muestra un ejemplo donde el elemento, que va "emergiendo" va siendo colocado en su lugar correcto, para ya no tomarse en cuenta en las siguientes pasadas se representó con mayor tamaño y más oscuro. Considerando que se ordena en forma ascendente.

Número de pasadas = n (elementos) - 1

Pasadas a detalle

5	5	5	5	0	0	0
6	6	6	**0**	5	5	5
9	9	**0**	6	6	6	6
4	**0**	9	9	9	9	**0**
0	4	4	4	4	**0**	9
0	0	0	0	0 1ª pasada	4	4

0	0	0	0	0	0	0
5	0	0	0	0	0	0
0	5	5	5	4	4	4

6	6	6	**4**	**5**	5	5
9	9	**4**	6	6	**6**	6
4	4	9	9	9	9	9
	2ª pasada			3ª pasada	4ª pasada	5ª pasada

Figura 2.3. Clasificación por el método de burbuja

Los números que van quedando con color marrón, son los que se acomodan en la posición que les corresponde.

Si se observa bien, se ve que desde la tercera pasada ya se encuentran acomodados en forma ascendente los elementos y no se realiza ningún intercambio, pero de cualquier forma es necesario realizar todas las comparaciones de todas las pasadas que restan.

El número de pasadas es igual al número de elementos menos uno.

2.2. Ordenamiento por inserción directa

Definición del método

Los métodos del grupo de Inserción, parten del supuesto, que el conjunto de llaves, ya se encuentra clasificado y cada vez que se desee agregar un nuevo elemento con una **llave K** determinada, bastará con agregarla a ese conjunto en la posición que le corresponda (esto se sabe a través de comparaciones y dependiendo si está en forma ascendente o descendente), haciendo los corrimientos de los elementos que sean necesarios. Así, el conjunto seguirá clasificado.

Este algoritmo considera que el conjunto de elementos sólo tiene un elemento y si tiene un solo elemento, el conjunto se encuentra clasificado, ya que el único elemento, es el que tiene tanto el valor más grande como el más chico.

En este conjunto, se irán insertando cada uno de los nuevos elementos y en su caso se realizaran los corrimientos correspondientes de los demás elementos del conjunto, para que siga quedando clasificado. A continuación se muestra en pseudocódigo este método:

n es el número de elementos con que cuenta el conjunto de elementos a ordenar y suponiendo que se desean clasificar en forma ascendente.

El pseudocódigo del algoritmo es el siguiente:

```
inicio
        j = 1
        hacer mientras j < n
                i = j – 1
                k = a[j]
                hacer mientras    ( i ≥ 0 )
                        si k < a[i] entonces
                                a[i+1] = a[i]          // se realiza el intercambio
                                a[i] = k
                        fin
                         i = i – 1
                fin
                j = j + 1
        fin
fin
```

En la séptima instrucción es donde se determina si se va a clasificar en forma ascendente (si k < a (i)) o descendentemente (si k > a(i)).

El elemento nuevo se va agregando a la derecha y se va comparando con el último elemento de los que están ya clasificados, si fuera en forma ascendente,

ese nuevo elemento se va recorriendo hacia la izquierda mientras encuentre elementos mayores o iguales a el.

Ejemplo

A continuación, se muestra un ejemplo con datos de este método, con el criterio de clasificación en forma ascendente:

(0)	(1)	(2)	(3)	(4)	(5)	(7) # dir
8	4	2	5	2	1	9 elementos a clasificar

Pasada 0	8	4	2	5	2	1	9
Pasada 1	4	8	2	5	2	1	9
	4	2	8	5	2	1	9
Pasada 2	2	4	8	5	2	1	9
Pasada 3	2	4	5	8	2	1	9
	2	4	5	2	8	1	9
	2	4	2	5	8	1	9
Pasada 4	2	2	4	5	8	1	9
	2	2	4	5	1	8	9
	2	2	4	1	5	8	9
	2	2	1	4	5	8	9
	2	1	2	4	5	8	9
Pasada 5	1	2	2	4	5	8	9
Pasada 6	1	2	2	4	5	8	9

Figura 2.4. Clasificación por el método de inserción directa

Los números marcados con azul son los que se van agregando al conjunto de elementos y se van acomodando, de acuerdo a su valor.

2.3. Ordenamiento por selección

Este grupo de métodos se basan en seleccionar del conjunto de datos el más pequeño o el más grande y separarlo del resto de los datos, haciendo nuevamente el procedimiento sobre los elementos restantes.

En este grupo los métodos más comunes son como se ve en la figura 2.1, son:

- Selección Directa
- Selección Repetitiva
- Torneo
- Heap

Ordenamiento por Selección Directa

Definición del método

Este algoritmo funciona buscando al elemento más pequeño (si fuera en orden ascendente o el más grande si fuera en forma descendente) en cuanto a valor de toda la lista de elementos originales, después se realizará el intercambiado por el que se encuentra en la primera posición de la lista, de esta forma queda colocado el elemento más pequeño en la posición que le corresponde, marcándolo de alguna manera para que ya no se tome en cuenta para las siguientes pasadas.

Para la segunda pasada, se vuelve a buscar al elemento más pequeño de la lista restante (sin considerar al que ya fue elegido) que es colocado en la primera posición de esta lista, nuevamente marcando este elemento, para que ya no sea tomado en cuenta para las pasadas que restan. Y así se seguirán realizando las diferentes pasadas, hasta que todos los elementos, queden clasificados.

En cada pasada se coloca un elemento a excepción de la última, en la cual se colocarán dos elementos, ya que al quedar sólo dos de ellos y escoger al más chico y colocarlo, el otro automáticamente queda colocado en su lugar.

El código del algoritmo es el siguiente:

```
void sel_directa(int a[], int num, int z)
{
   static int x;
   int c=0,i=0,k,j=0;
   printf("\n\t**** METODO DE SELECCION DIRECTA *****\n\n");

   printf("\n\t*** los elementos desordenados son ***\n");
   for (k=0;k < num; k++)
          printf (" %d ",a[k]);
   printf("\n\n");
   num=num-1;
/* Ordenamiento por seleccion directa */
   principio = clock();
   for(i=0;i < num;i++)   // i apunta a el primer elemento de la lista
   {                    //que queda para comparar
        delay(100);
        k=i;
        for (j=i+1; j <= num; j++)
        {
           if (a[j] < a[k])  //< es descendentemente y > ascen.
              k=j;           //k apunta al elemento mas chico
        }
        if (i!=k)            //si i <> k se hace el intercambio y se
pasa
        {                    //el mas chico (k) a la posicion de i
              c=a[i];
              a[i]=a[k];
              a[k]=c;
         }
    }
    fin = clock();
```

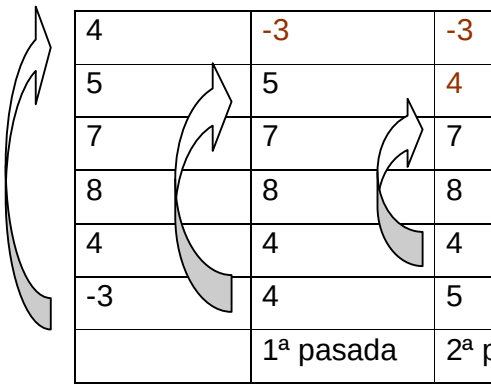
```
    printf("\n\t*** los elementos ordenados en forma");
    printf(" asecendente son ***\n");
    for (k=0;k <= num; k++)
          printf (" %d ",a50[k]);
   nele[0][z]=num+1;
   ntiempo[0][z]=((fin - principio)) / CLK_TCK ;
   printf("\n\n");
   printf("\n\t*** El tiempo de ejecucion en decimas de segundos\n");
   printf("\t\ten forma ascendente fue de: %7.5f ",((fin - principio))
/ CLK_TCK);
   printf("****");
   getchar();
} // fin de sel_directa
```

Una salida de este programa es la siguiente:

```
       **** METODO DE SELECCION DIRECTA *****
       *** los elementos desordenados son ***
 62  39  62  77  77  86  74  55  82  86  3  88  44  40  9


       *** los elementos ordenados en forma ascendente son ***
 3  9  39  40  44  55  62  62  74  77  77  82  86  86  88

       *** El tiempo de ejecución en décimas de segundos
              en forma ascendente fue de: 0.16484 ****
```

Ejemplo

En la siguiente figura, se presenta un breve resumen de cómo funciona el método:



4	-3	-3	-3	-3	-3
5	5	4	4	4	4
7	7	7	4	4	4
8	8	8	8	5	5
4	4	4	7	7	7
-3	4	5	5	8	8
	1ª pasada	2ª pasada	3ª pasada	4ª pasada	5ª pasada

Figura 2.5. Clasificación por el método de selección directa

No. de pasadas = elementos – 1 es (n-1) sería para este ejemplo (6-1)

Los elementos que van quedando marcados y que ya no serán considerados en la siguientes pasadas, se muestran en las posiciones de arriba con números más grandes y remarcados con color marrón.
La flecha indica el intercambio de elementos.

Ordenamiento por Selección Repetitiva

Definición del método

Este método, trata de optimizar el de Selección Directa, cuando utilice un mayor número de datos, ya que va a dividir el conjunto de datos a clasificar en pequeños grupos y cada uno de estos grupos va a obtener el mayor (si está clasificado en forma descendente) o menor elemento (si es de forma ascendente), ese elemento, va ha ser el "ganador" de su grupo y pasará a otra etapa donde "competirá" con los "ganadores" de los otros grupos y el nuevo "ganador" (más chico o más grande según sea el caso), será el elemento que saldrá del conjunto de datos y se colocará en otro arreglo (o archivo), en la primera posición.

De donde salió este último "ganador", se ve de que grupo fue y se le indica a ese grupo que mande otro representante o "ganador", al grupo de "ganadores"

para que vuelva a competir y nuevamente saldrá un “ganador” de todos los grupos y este ingresa a la segunda posición del arreglo donde se van colocando los elementos, ya clasificados y así se sigue, hasta que todos los elementos, ya han sido transportados al nuevo arreglo.

Cuando alguno de los grupos originales, ya no tenga algún elemento a enviar, esto es ya no tenga ningún “jugador”, es porque ya todos salieron y se irá marcando de alguna manera, para indicar, que ya no tiene elementos, podría ser por ejemplo con **infinito** ∞, el cuál será un valor muy grande positivo, si se trata de una clasificación ascendente o **menos infinito** $-\infty$, si es la clasificación descendente.

Para el tamaño de los nuevos grupos, se pueden tomar varios criterios como el de la raíz cuadrada $\sqrt{N}$, donde **N,** es el número de elementos que se desean clasificar, por ejemplo si se desean clasificar **N =** 16 elementos, la $\sqrt{16}$ = 4, entonces los grupos serán 4 y cada grupo tendrá 4 elementos y el grupo de “ganadores”, también será de 4. Esto es el resultado de la raíz cuadrada, indica el tamaño de los grupos, el número de grupos y el tamaño del grupo de ganadores.

En este método se va a mostrar su pseudocódigo, suponiendo que estarán clasificados en forma ascendente.

Los **arreglos** que se utilizan son:

- datos de una dimensión, donde se almacenarán los datos a clasificar y que originalmente están desordenados.

- **ganadores,** de dos dimensiones, donde la primera columna representa el dato (llave o clave) más pequeño y se va al grupo de ganadores.

- **menor** que es un arreglo de una sola dimensión y va a almacenar de uno en uno los elementos que van “ganando” en cada pasada y es donde al final del método, van a quedar clasificados en forma ascendente.

El Pseudocódigo del algoritmo es el siguiente:

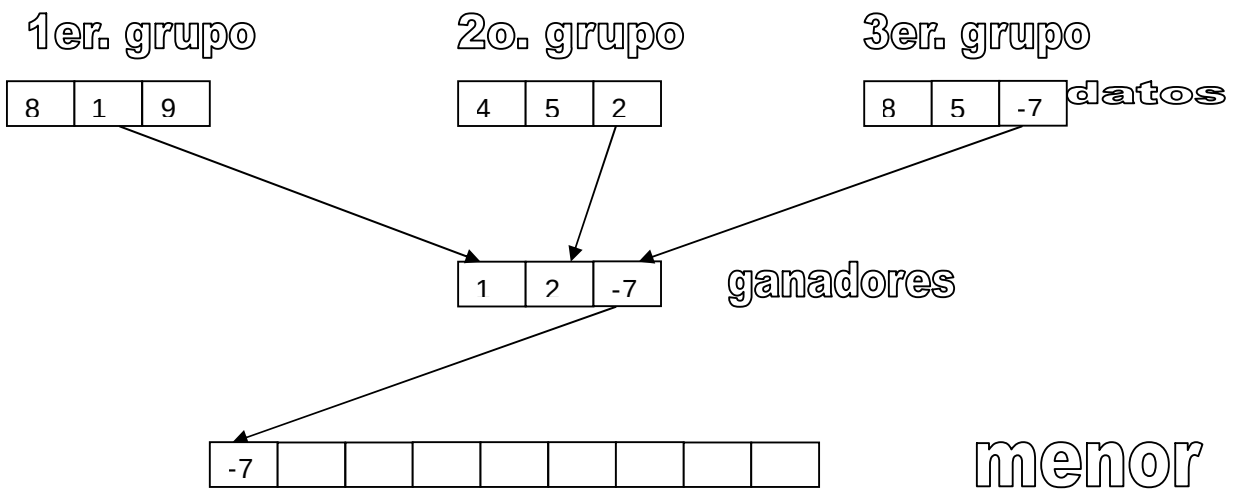
```
inicio
        indice= - 1
        i = √n
        p = -1
        k = -i
        j = 0
        hacer mientras ( j <  i  )
                p = p + 1
                k = k + i
                m = k
                c = k +1
                hacer mientras ( c ≤  ( k + i – 1 )  )
                        si datos [c] < datos [m] entonces
                                m = c
                        fin
                        c = c + 1
                fin
                j = j +1
                ganadores [p][0] = datos [m]
                ganadores [p][1] = m
        fin

        j = 0
        hacer mientras ( j <  n  )
                m = 0
                c =  m +1
                hacer mientras ( c < i  ) // se seleccionan el ganador de los
ganad.
                        si ganadores [c][0] < ganadores [m][0]
                                m = c
```

```
                    fin
                    c = c + 1
                fin
                p = m // p dice de que grupo salio el ganador
                indice = indice + 1
                menor [indice] = ganadores [m][0]   // es el ganador de todos
                datos [ ganadores [m][1] ] = infinito
                k= i * m
                m = k
                c= m +1
                hacer mientras ( c ≤  ( k + i – 1 ) )
                    si datos [c]  < datos [m]
                        m = c
                    fin
                    c = c + 1
                fin
                ganadores[p][0]=datos[m]
                ganadores [p] [1] = m
                j = j + 1
        fin
fin
```

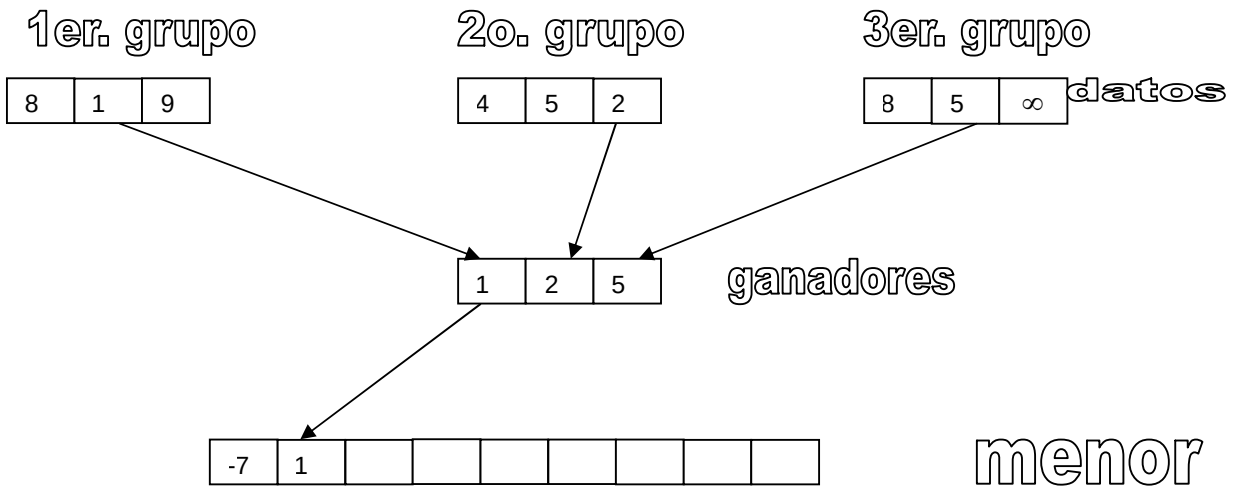
En el arreglo **menor** se van a tener los elementos ya clasificados en forma ascendente.

Ejemplo

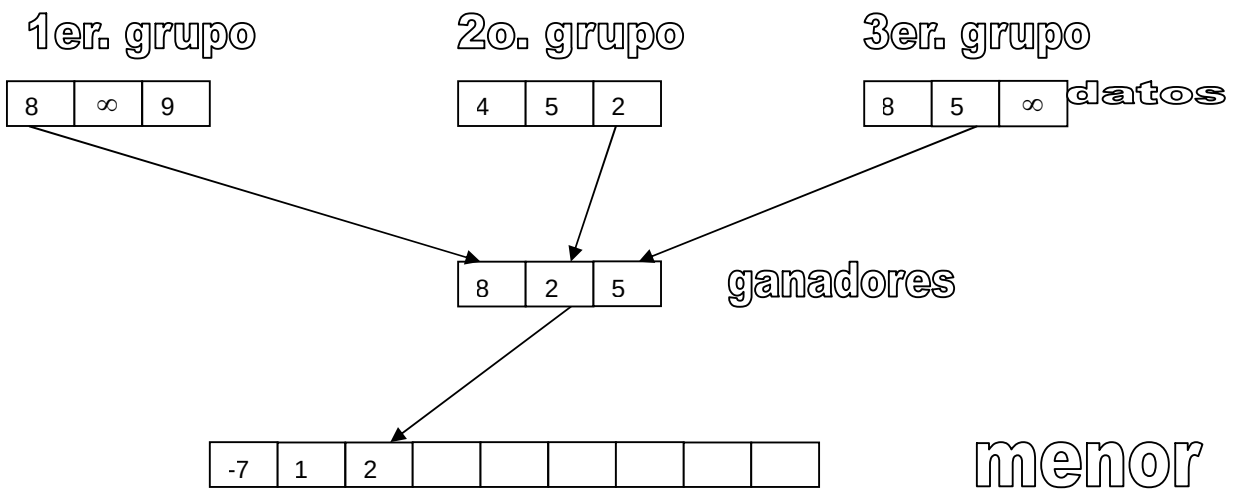
A continuación se ejemplifica el método con un conjunto de elementos suponiendo que están clasificados en orden ascendente:



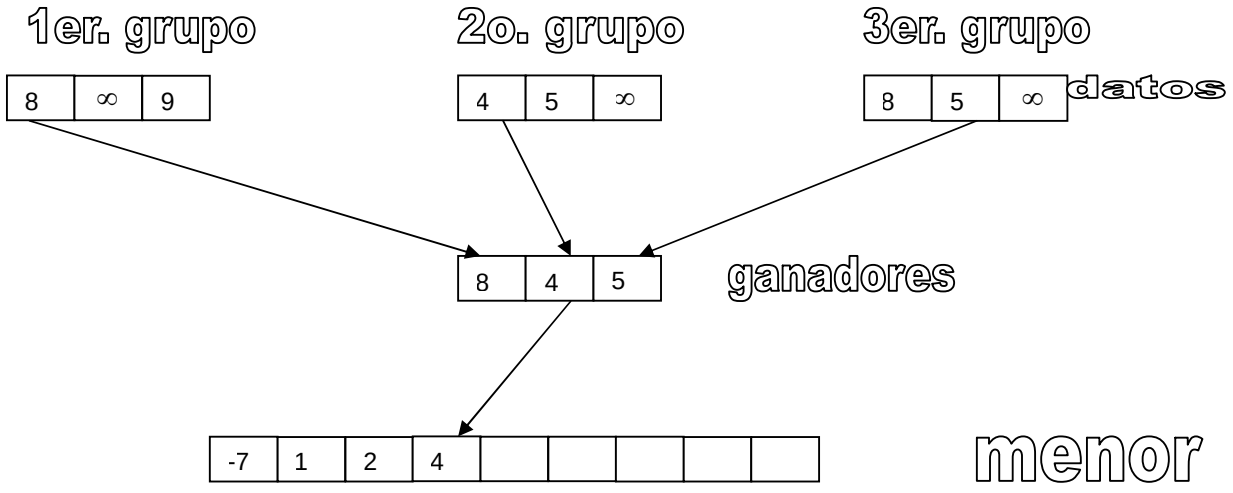
Primera pasada



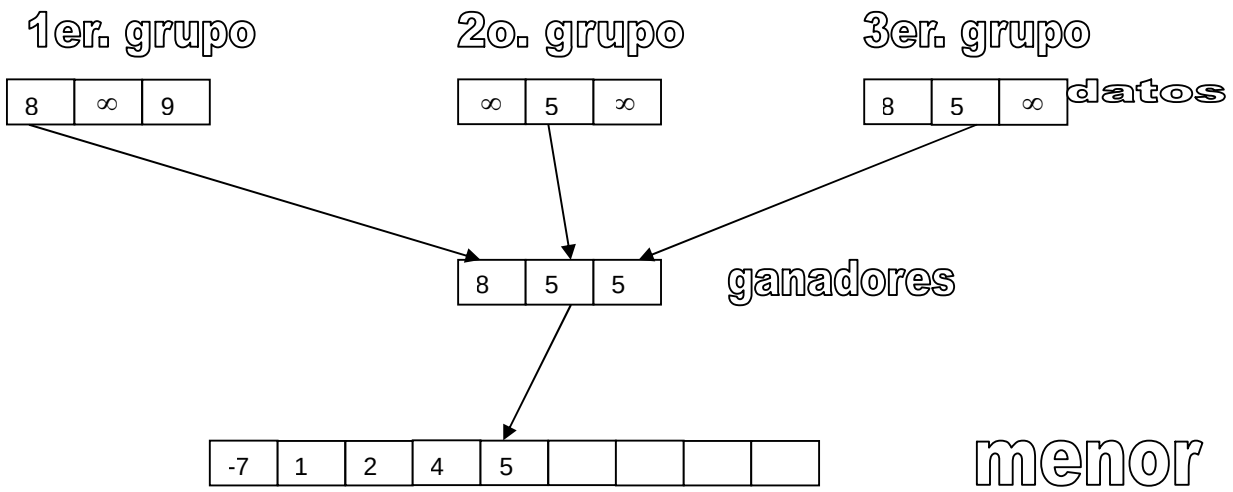
Segunda pasada



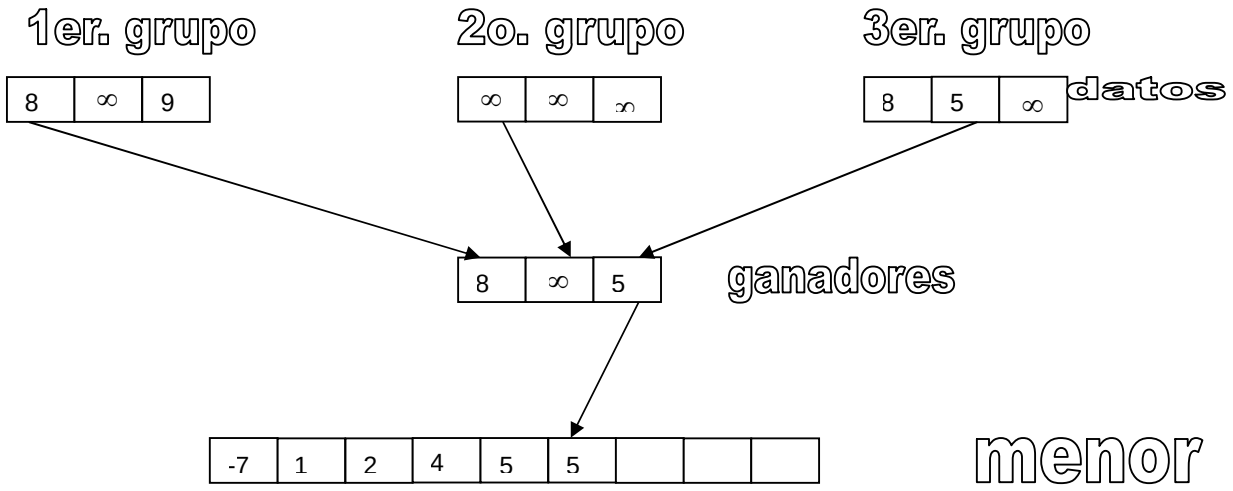
Tercera pasada



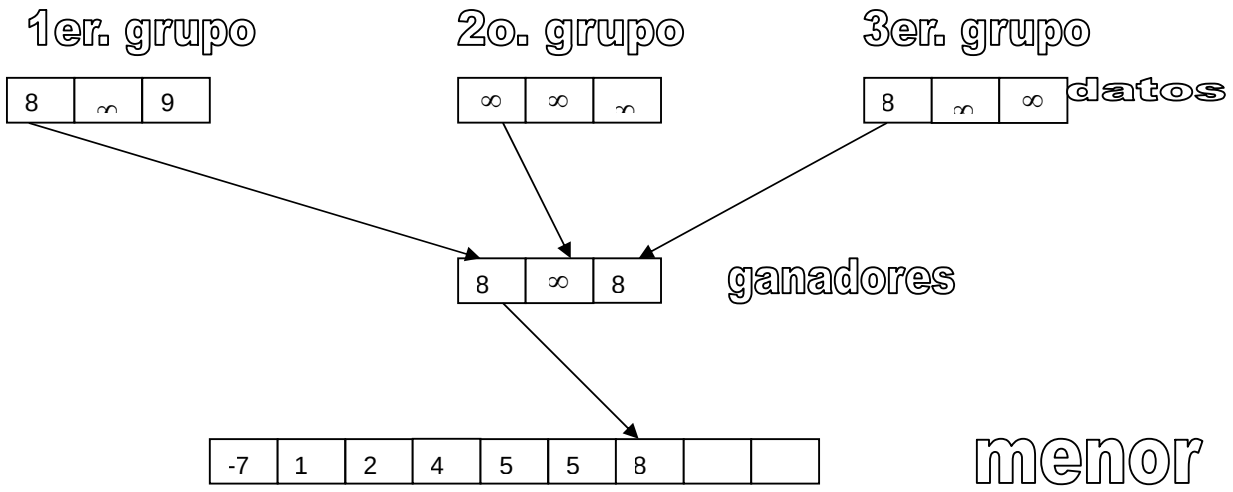
Cuarta pasada



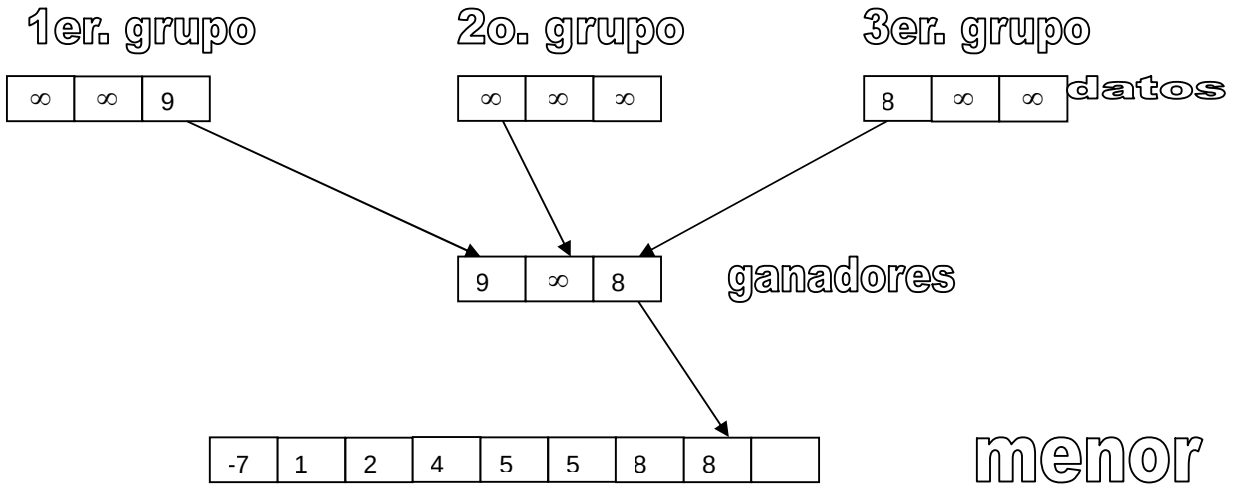
Quinta pasada



Sexta pasada

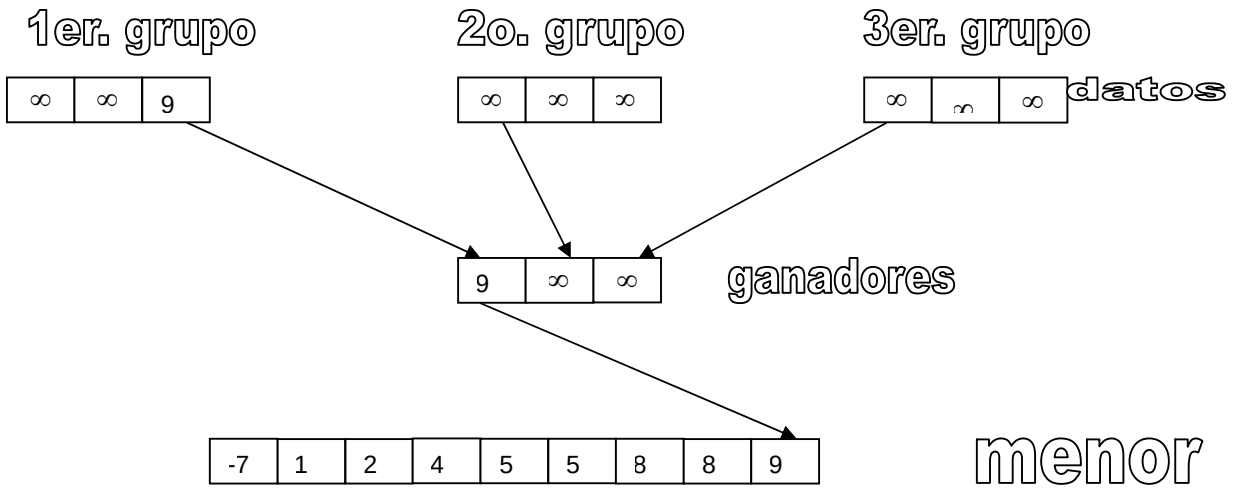


Séptima pasada



Octava pasada

∞



Novena pasada

Figura 2.6. Clasificación por el método de selección Repetitiva.

Finalmente, los arreglos quedarían de la siguiente forma:

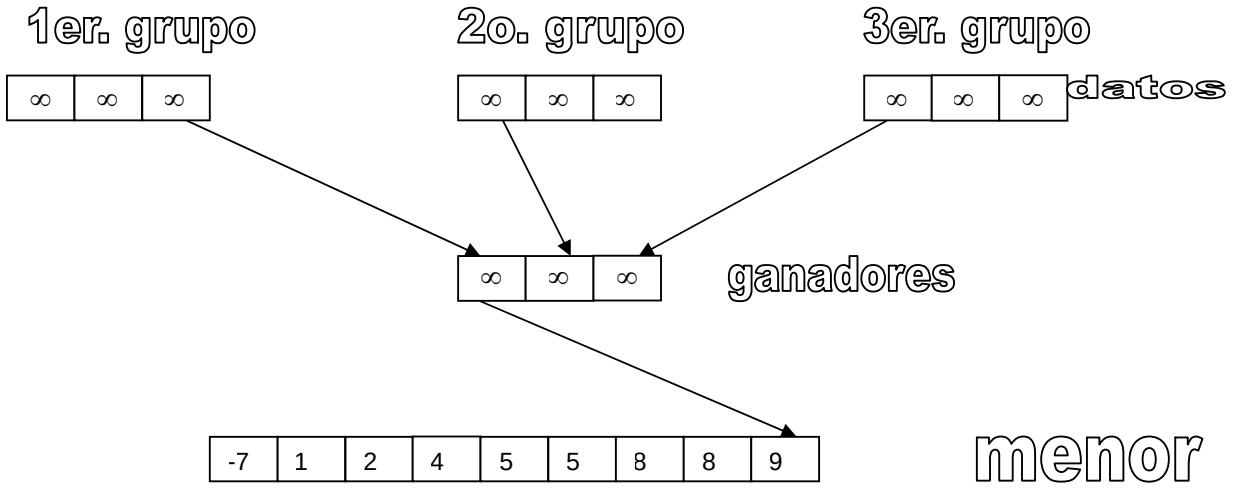


Figura 2.7. Etapa final de la clasificación por el método de selección directa

Pero ¿qué pasa si el número de elementos no es exacto a la raíz cuadra? Una posible solución sería tomar el número entero siguiente y completar los

elementos de los últimos grupos como marcados de que ya salieron con infinito ∞.

Por ejemplo, si se tienen 10 elementos, entonces $\sqrt{10}$ = 3.1622 y los grupos serían de 4.

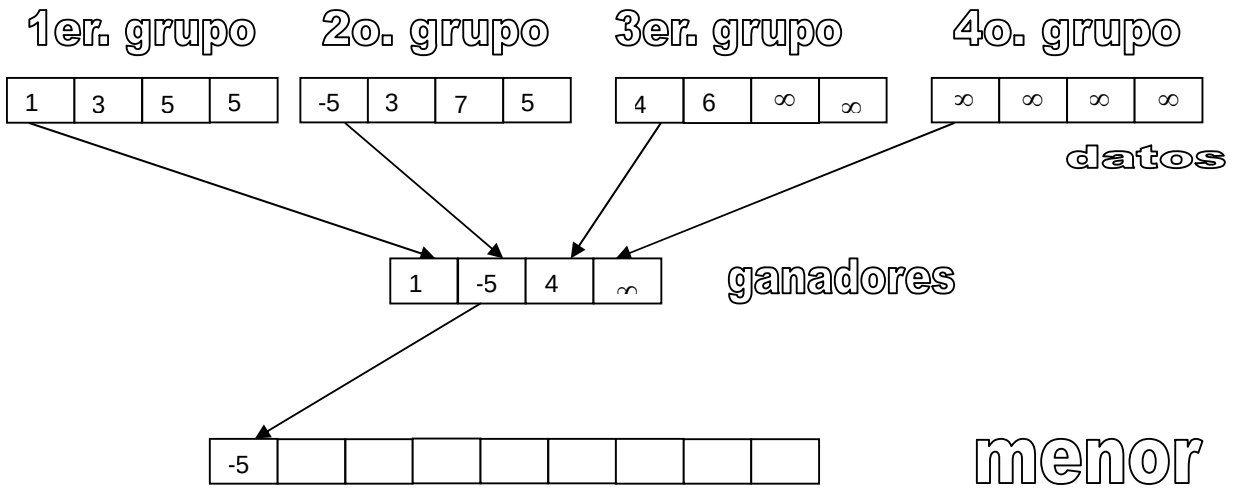


Figura 2.8. Método de selección directa cuando la raíz cuadrada del número de elementos, no es exacta.

Para terminarlo sería el mismo procedimiento visto en el ejemplo anterior, lo único que varia es que desde la primera pasada ya vienen elementos marcados con infinito (∞), como si ya hubieran sido utilizados. Nótese que el algoritmo que se describió en pseudocódigo, no contempla este caso.

Se puede tener más variantes de este método, ya que en lugar de raíz cuadrada puede ser cúbica, cuarta, quinta, etc.

Si se observan las pasadas, se verá que se va comportando como un árbol invertido, donde la raíz del árbol es donde se encuentran los “ganadores”, el número de la raíz de **n**, indica el tamaño de cada grupo, así cómo cuántos niveles va a tener el árbol. Cada nivel indica otra etapa de eliminación. Es de hacer notar que aparte se maneja el arreglo llamando **menor**, donde se van colocando ya todos los elementos “ganadores”, según van saliendo. Haciendo

una analogía con un campeonato de fútbol, se tienen octavos y cuartos de final, semifinal y final, por ejemplo si se tiene:

$\sqrt[2]{n}$, el número de niveles es de 2 (la raíz del árbol y los nodos terminales), se tienen en un nivel todos los datos y en otro nivel se tienen a los "ganadores", como el ejemplo anterior visto de este método.

$\sqrt[3]{n}$, el número de niveles es de 3 (la raíz del árbol y dos niveles más), que indican; el nivel donde esta todos los datos, otro nivel de semifinal y un tercer nivel de "ganadores", de la gran final.

$\sqrt[4]{n}$, el número de niveles es de 4 (la raíz del árbol y tres niveles más), los cuales tendrán en el primer nivel, todos los datos, en un segundo nivel los números "ganadores" de cuartos de final, un tercer nivel los elementos "ganadores" de la semifinal y cuarto nivel con los "ganadores de la gran final. Para el resto de las raíces, quinta, sexta, etc., el árbol que se va ocupando va ir creciendo.

A continuación se ejemplifica para el caso de raíz cúbica de **n** ($\sqrt[3]{n}$), como quedaría el árbol de tres niveles, suponiendo que **n** es de 27 elementos, entonces $\sqrt[3]{27}$ = **3**, entonces serán 9 grupos de **3** elementos y **3** niveles del árbol. Salieron 9 grupos, ya que si el tamaño del grupo fue de **3**, para tener los 27 elementos, se requiere multiplicar por 9.

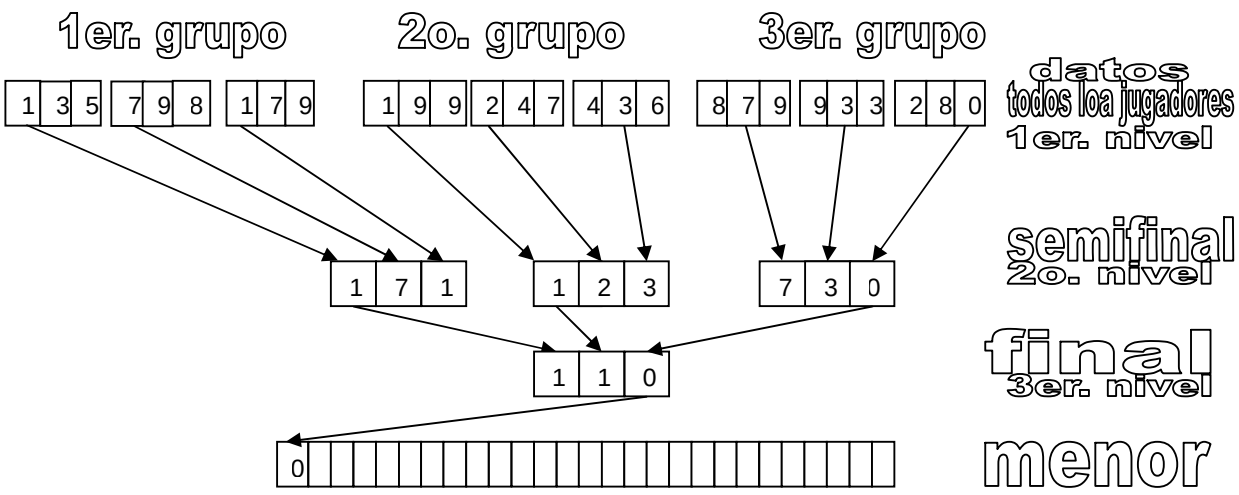


Figura 2.9. Primera pasada del método de Selección Directa con $\sqrt[3]{n}$

Torneo

Definición del método

El método de **torneo** es una variante de **selección repetitiva**, que se da cuando sólo dos elementos "compiten" entre sí, su nombre se debe a los torneos medievales donde los "caballeros medievales", competían de dos en dos y el ganador era el que pasaba a siguiente etapa de eliminación por lo que sólo llegaban la mitad de los caballeros y según se iban desarrollando cada una de las etapas, al final queda únicamente uno de los caballeros como ganador único.

En el caso de este método de clasificación los "caballeros medievales" son los datos o llaves a clasificar.

Ejemplo

En el siguiente ejemplo, sólo se mostrará la pasada, donde se saca al primer "ganador", ya que prácticamente el método es igual a los descritos de este grupo con anterioridad.

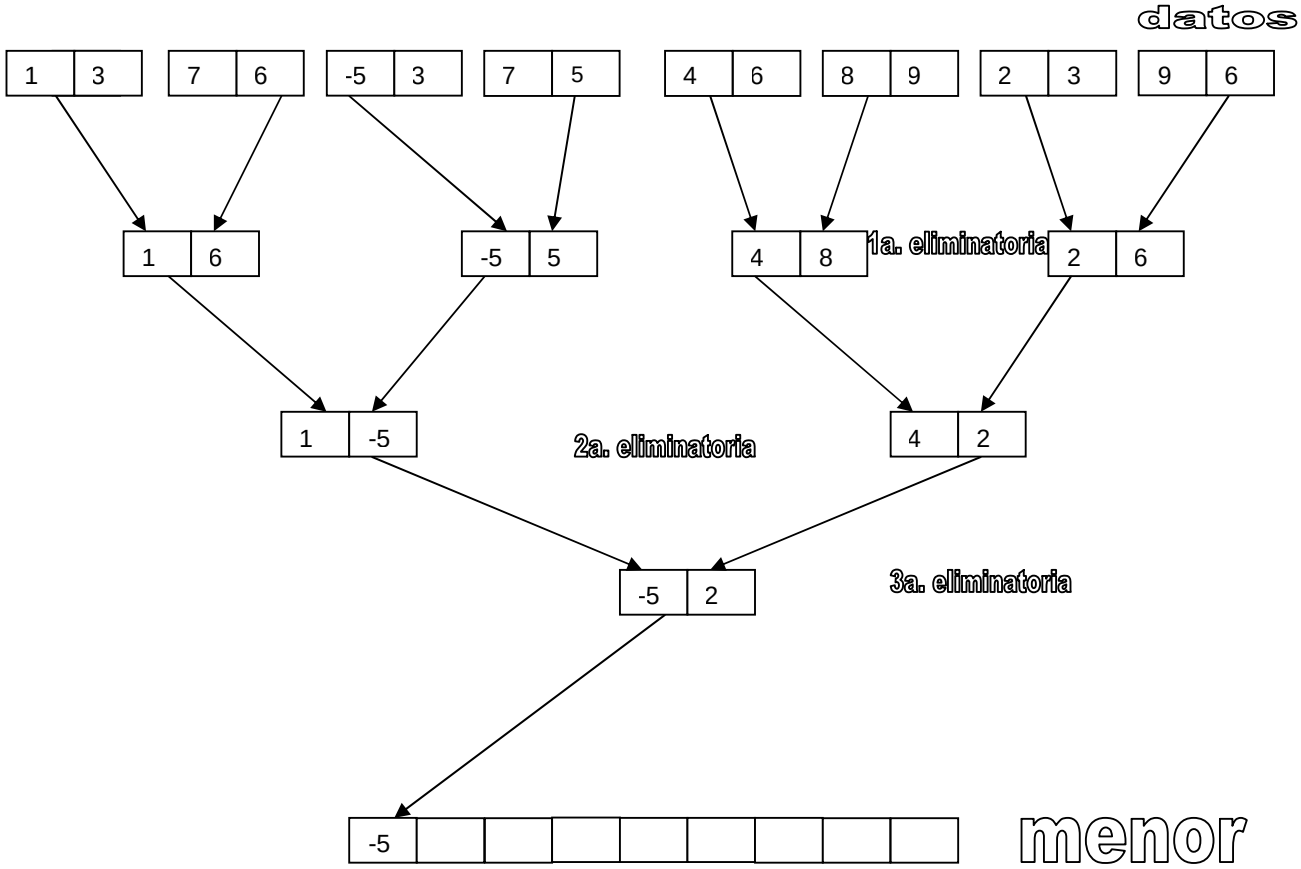


Figura 2.10. Primera pasada del método de Torneo.

Heap

Definición del método

Este método consiste en la utilización de un árbol binario, el cual va a estar constituido de uno o varios subárboles.

El método va a constituir de varios **pasos**, que son los que a continuación se describen:

1. Creación del árbol binario.- Se crea un árbol binario, con los elementos que se van a clasificar, como si fuera el resultado de haber recorrido el árbol de arriba-abajo (*top-down*).
2. División en subárboles binarios.- Este árbol se va a ir descomponiendo en subárboles binarios, pero sólo uno a la vez, partiendo del nivel mas bajo y siguiendo la secuencia de derecha

a izquierda. Estos subárboles serán de tres elementos, solo en el nivel más bajo podría darse el caso de tener un subárbol de dos elementos.

3. Reacomodo de los valores de los nodos.- En el subárbol que se encuentra en el nivel más bajo y más a la derecha del árbol, si es necesario se realizan los movimientos para cambiar de posiciones los elementos de los nodos, de tal forma que, el nodo raíz de cada subárbol quede con un valor mayor o igual a cualquiera de sus ramas (ascendentemente), o un valor menor o igual a cualquiera de sus ramas (descendentemente).
4. Creación de la estructura del Heap.- Este procedimiento se va haciendo, en cada subárbol, siguiendo la secuencia de derecha a izquierda y hacia arriba, hasta llegar a la raíz principal de todo el árbol, cuando esto sucede quiere decir que en la **raíz principal** del árbol se tendrá al elemento más grande (ascendente) o más chico (descendente) de todo el árbol. A esta estructura árboles le conoce como una estructura de ***Heap*.**
5. Intercambio del dato de la raíz.- El siguiente paso consiste en realizar un intercambio del elemento que se encuentra en el nivel más abajo y más a la derecha, de todo el árbol, por el elemento que se encuentra en su raíz principal, una vez hecho este intercambio, se marca de alguna forma el elemento que llegó al nivel más bajo, con lo que termina cada pasada.

En la siguiente pasada, ya no se toma en cuenta el elemento que se marcó y se realizan nuevamente los pasos del 2 al 5, descritos anteriormente, con el resto de los elementos, nuevamente se crea la estructura de **H*eap*** y se realiza el siguiente intercambio y así se continuamente las siguientes pasadas, hasta que sólo queden dos elementos (un nodo y la raíz) sin marcar.

En cada pasada se van colocando un elemento, a excepción de la última, en donde se acomodan dos, por lo que el número de pasadas es de **n-1**, considerando que **n** es el número de elementos, que se van a clasificar.

Ejemplo

Supóngase que se desean clasificar los siguientes datos en forma **descendente**:

1, 9. 6, 7, 0, 2, 1, 3, 4, 8

El árbol que se creará es, siguiendo el criterio de que esta lista de datos fue el resultado de haber recorrido el árbol de arriba-abajo *(top-down*), de la siguiente forma:

Paso 1: Creación del árbol binario.

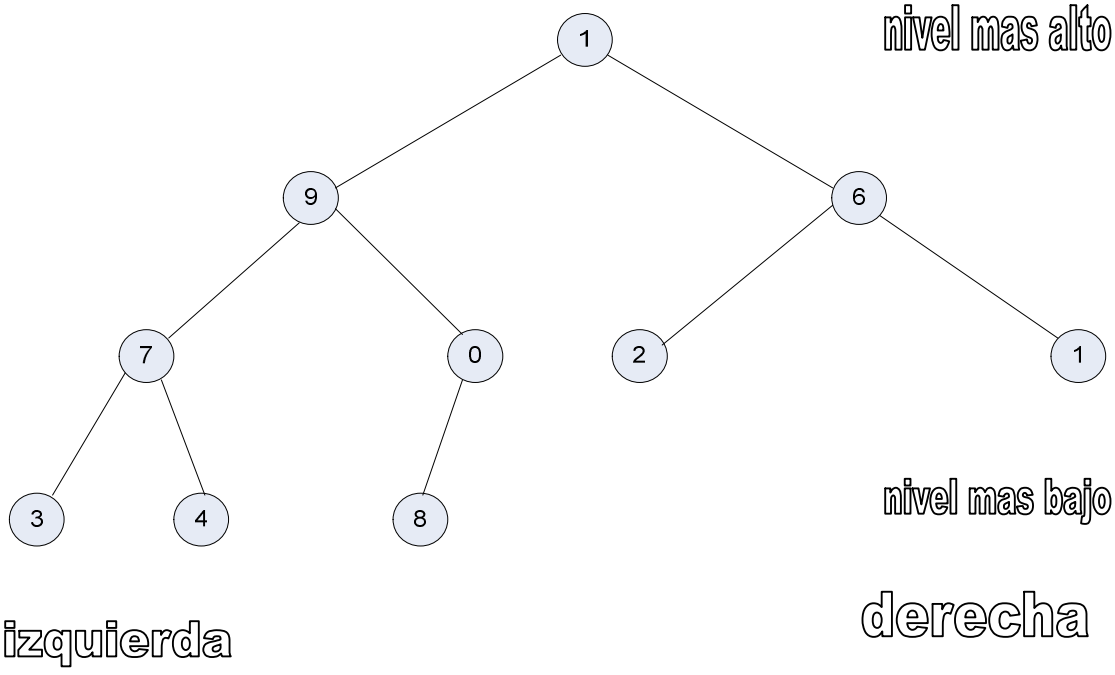


Figura 2.11. Árbol inicial del método de Heap.

Paso 2: División en subárboles binarios.

Se va dividiendo en subárboles binarios, empezando en el nivel más bajo y en la dirección de derecha a izquierda, como se muestra a continuación:

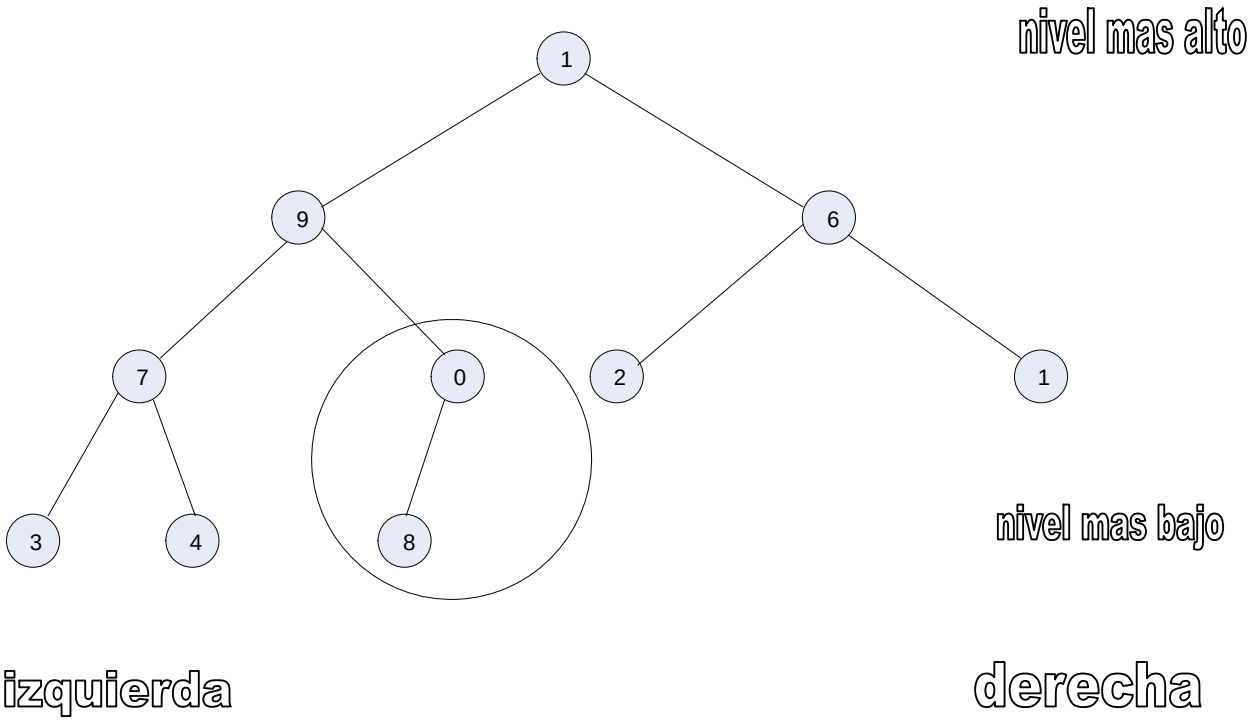


Figura 2.12. Primer subárbol binario del método de Heap.

Para aspectos didácticos, estos subárboles se encuentran encerrados en un círculo. Es importante destacar que este subárbol, sólo cuenta con dos nodos, ya que falta su rama derecha, pero sigue siendo binario.

Paso 3: Reacomodo de los valores de los nodos.

El subárbol que se encuentra encerrado en el círculo es el siguiente:

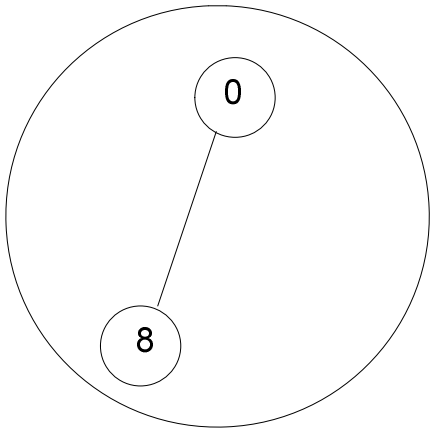


Figura 2.13. Análisis del primer subárbol binario del método de Heap.

La clasificación es en forma **descendente**, para los elementos de este ejemplo, por lo que en su raíz debe quedar un elemento con valor menor o igual a cualquiera de sus ramas, queda sin modificación, ya que el **cero**, es menor a **8**, que es su rama izquierda.
El árbol queda hasta este paso idéntico al de la Fig. 2.12.

Paso 4: Creación de la estructura del Heap.
Posteriormente, se siguen comparando el siguiente subárbol, siguiendo el mismo criterio, del nivel más bajo y más a la derecha. A continuación, se muestra la siguiente comparación.

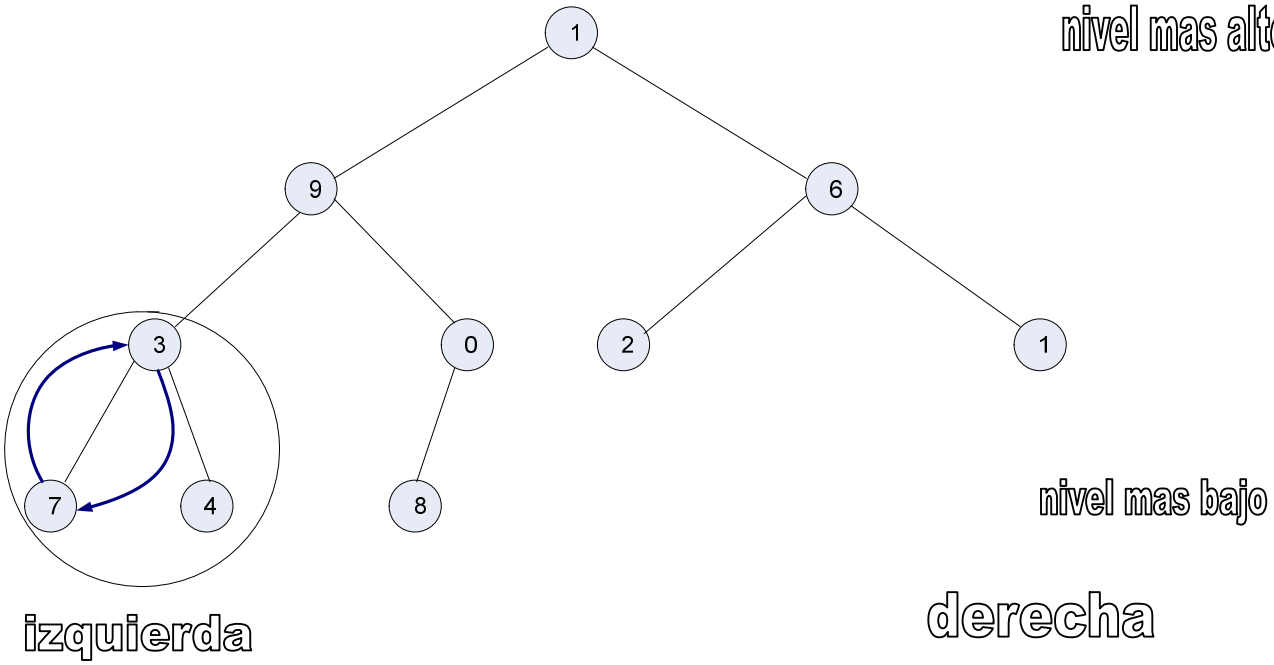


Figura 2.14. Análisis del segundo subárbol binario del método de Heap.

Aquí si se realiza un intercambio, ya que el valor de la llave más pequeño es el **3**, que se encontraba en la rama izquierda y sube a la raíz del subárbol, bajando el **7**, que originalmente se encontraba en esa raíz.

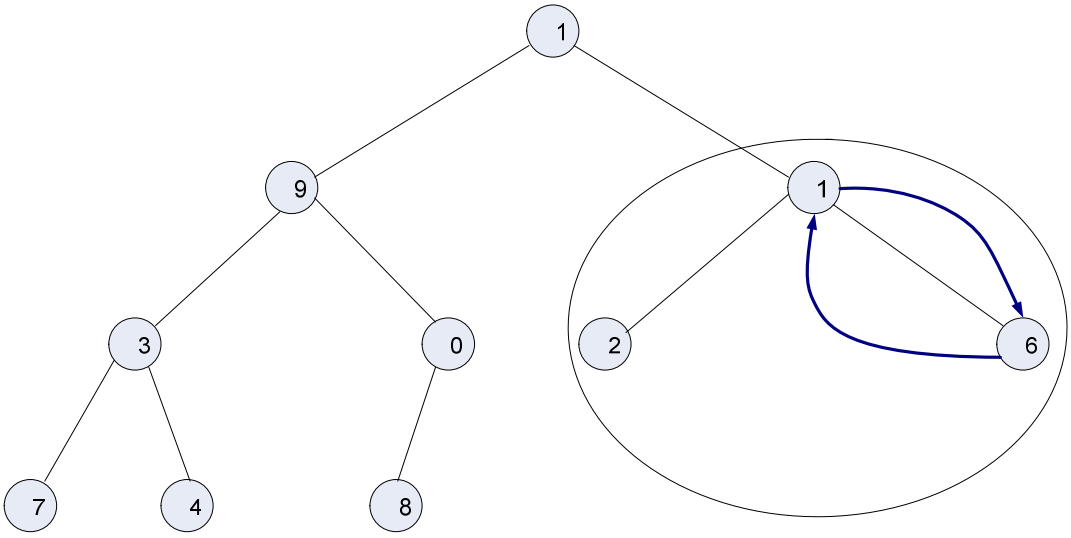


Figura 2.15. Análisis del tercer subárbol binario del método de Heap.

En esta comparación se intercambia el valor de la raíz por el de su rama derecha, en este caso el **1** por el **6**.

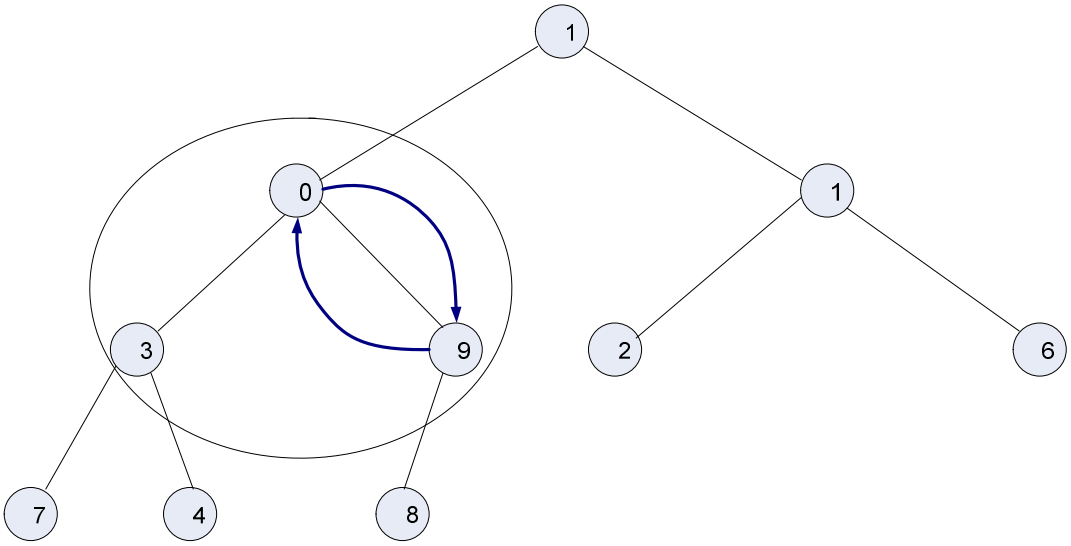
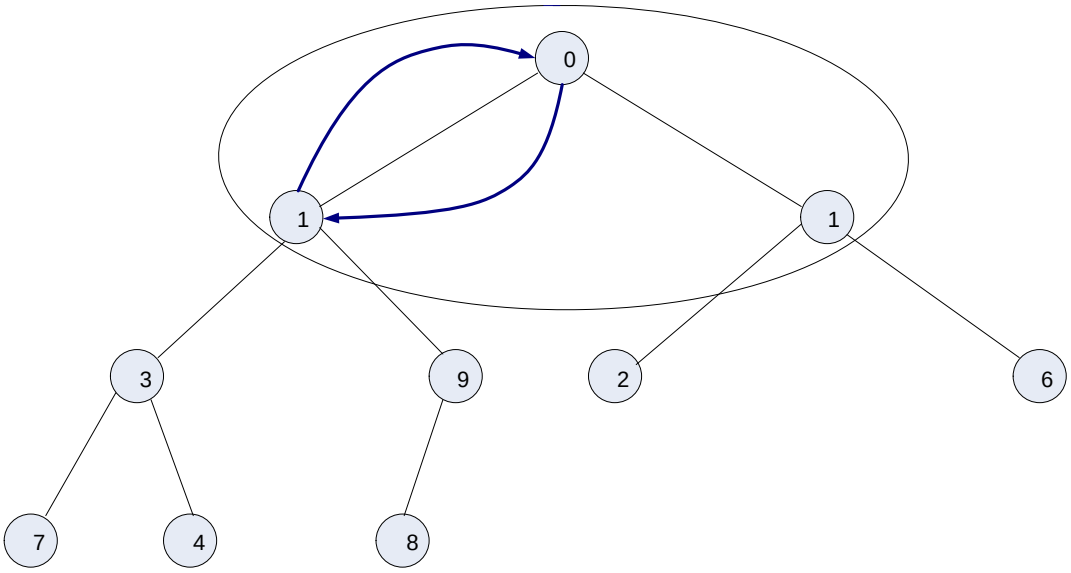


Figura 2.16. Análisis del cuarto subárbol binario del método de Heap.

En este subárbol se intercambia el dato de la raíz por el de su rama izquierda, en este ejemplo el **9** pasó al lugar del **cero**, en las raíces de cada subárbol va quedando el elemento más pequeño o igual al de sus nodos hijos (conocidos también como descendientes o ramas).



PRIMER HEAP

Figura 2.17. Análisis del quinto subárbol binario y primer Heap.

En este subárbol se intercambia el dato rama izquierda, por el de su raíz, para este caso es el **1** pasó al lugar del **cero**.

Primer *heap*

En esta parte de las comparaciones, cuando se llega a la raíz principal del árbol, se asegura que en esa raíz se encuentra el elemento más pequeño de todos los que se encuentran en el árbol, ya que en este ejemplo se está considerando una clasificación descendente (o el más grande si se trata de clasificar en forma ascendente).

Si se analiza, se verá que cualquiera de los subárboles tiene un elemento en su raíz, **menor o igual a cualquiera** de sus descendientes, entonces se considera que esta característica cumple con la estructura del **H*eap*,** que es el primer **Heap (**nombre recibe esta estructura) y que también es la mitad de la primera pasada.

Paso 5: Intercambio del dato de la raíz.

La otra mitad es el intercambio del elemento de la raíz por el que se encuentre más bajo en cuanto a niveles y más a la derecha marcándose de alguna forma ese elemento que se fue al nivel más bajo, para que no se vuelva a tomar en cuenta en la siguiente pasada, ya que ocupa su lugar dentro del árbol.

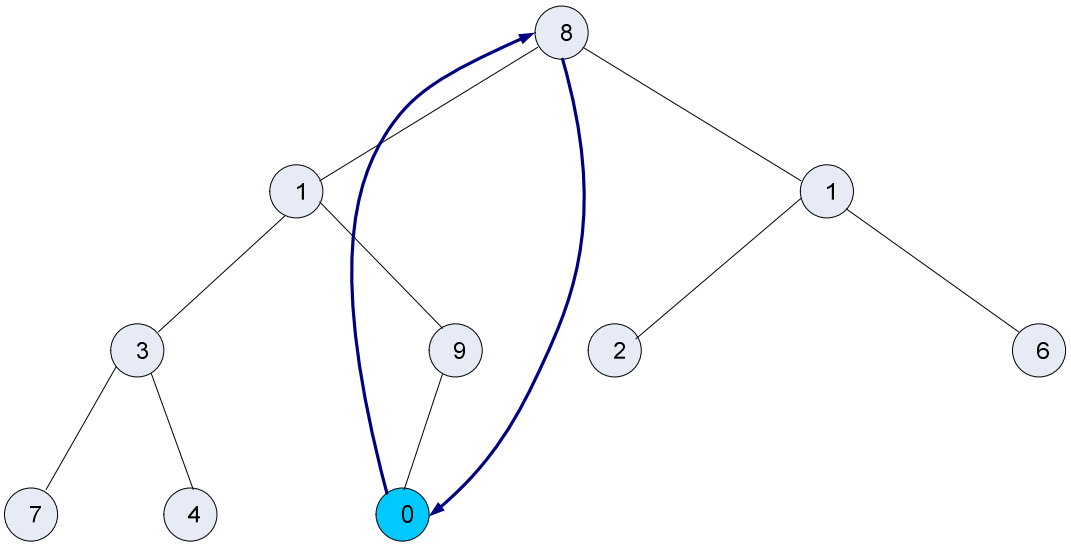


Figura 2.18. Fin de la primera pasada del método de Heap.

Se cambia el elemento de la raíz (más pequeño de todos), por el que está en el nivel más bajo y se marca, en este ejemplo se realizó con un color **azul cielo**.

Con el resto de los elementos, nuevamente, se irán comparando los subárboles del nivel más bajo y de derecha a izquierda, hasta llegar a la raíz principal, para crear la estructura del **H*eap***, y realizar el siguiente intercambio. Este procedimiento se realizará en cada pasada. A continuación, se ejemplifican los intercambios de la segunda pasada.

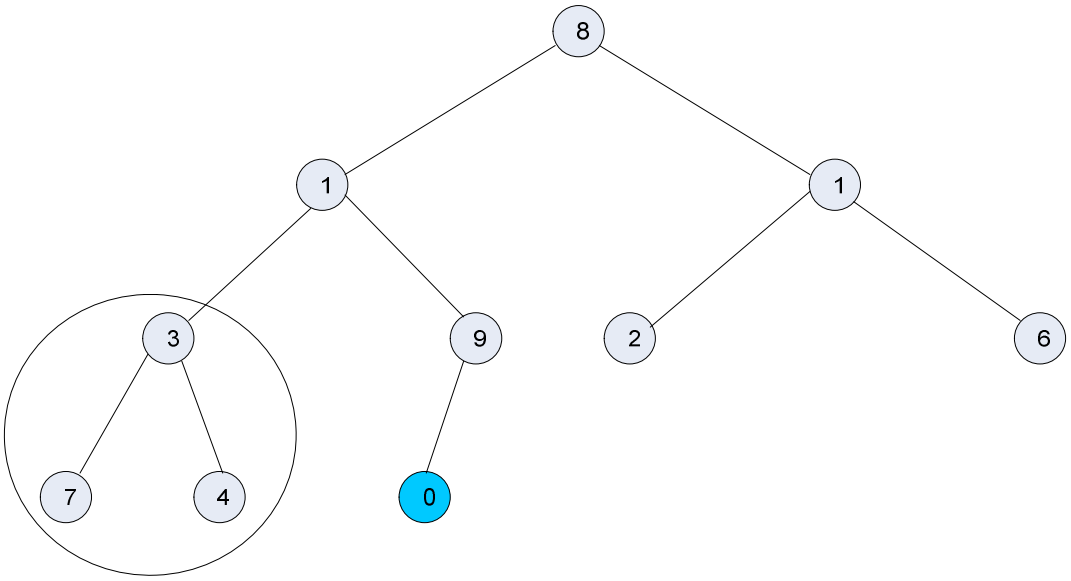


Figura 2.19. Análisis del primer subárbol binario, sin contar ya el elemento marcado.

En esta comparación no se realiza ningún intercambio, ya que va cumpliendo la estructura del **Heap**.

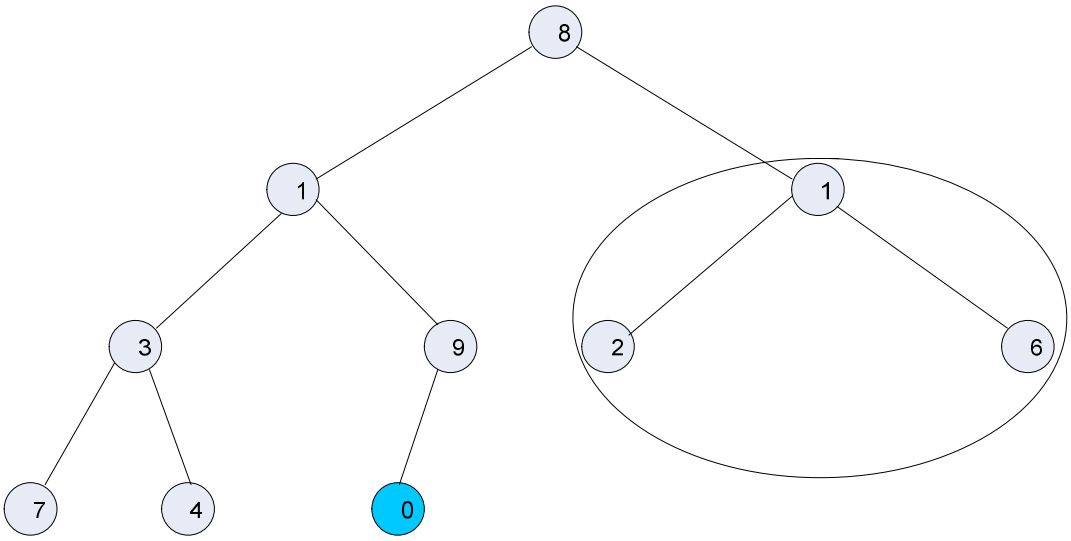


Figura 2.20. Análisis del segundo subárbol binario.

Tampoco se realizó ningún intercambio, pero se realiza la comparación.

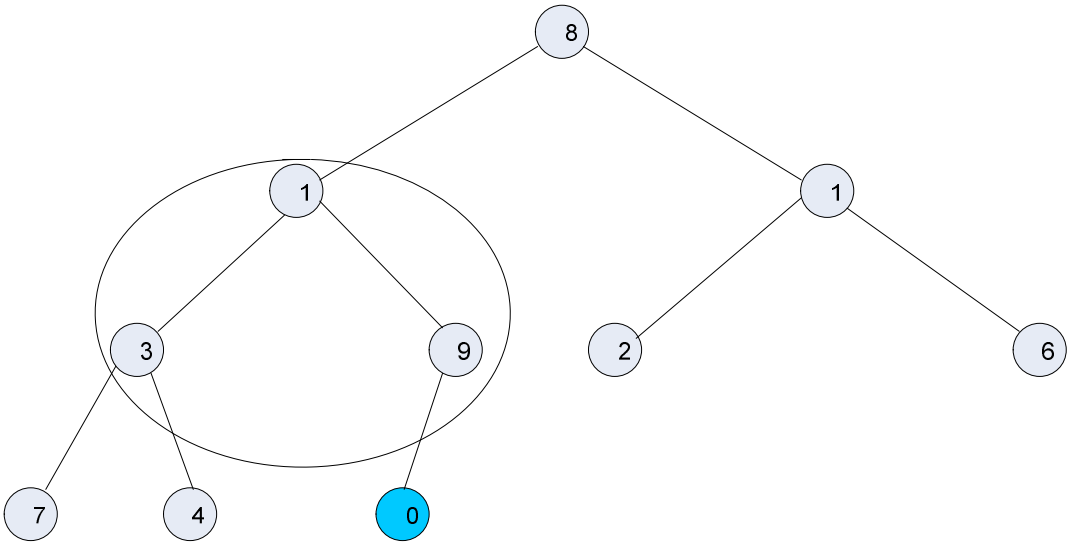
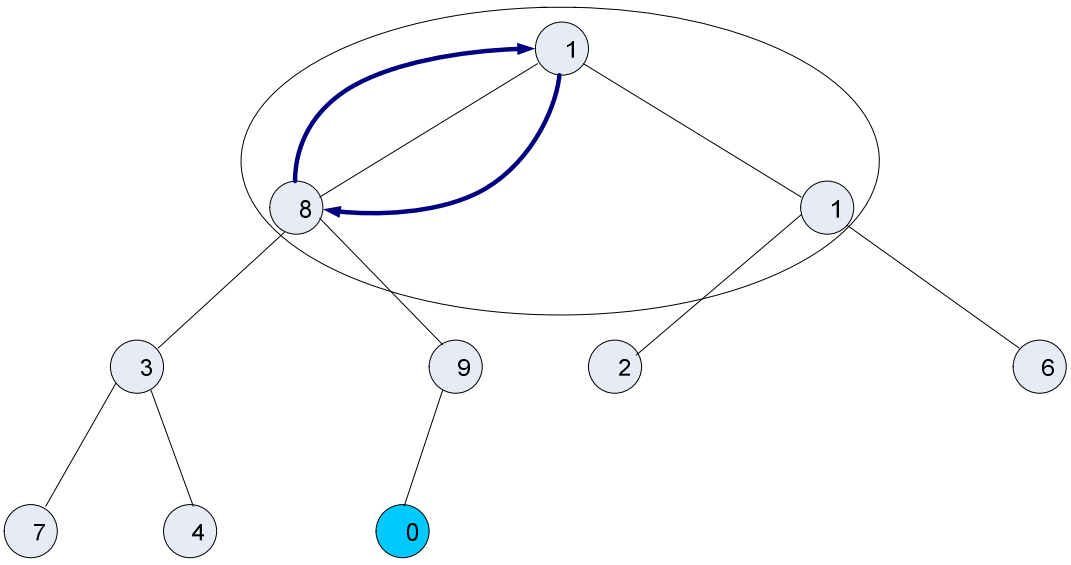


Figura 2.21. Análisis del tercer subárbol binario.

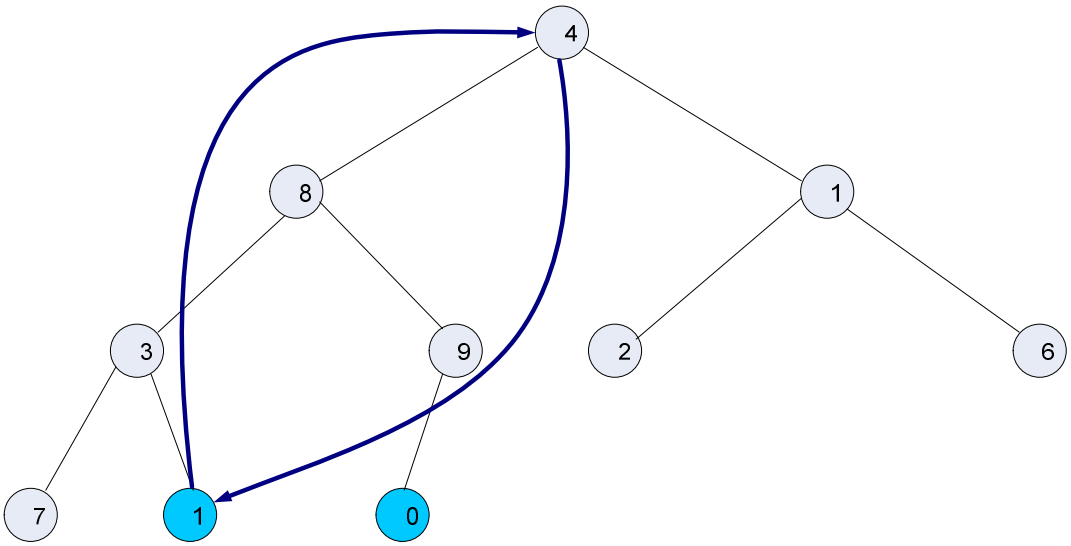
Nuevamente, no se realiza ningún intercambio.



SEGUNDO HEAP

Figura 2.22. Análisis del cuarto subárbol binario y segundo Heap.

Se realiza el intercambio de la raíz principal del árbol, por su rama izquierda, En este ejemplo se intercambia el **8** por el **1** y se obtiene la segunda estructura del **H*eap***.



SEGUNDO INTERCAMBIO

Figura 2.23. Fin de la segunda pasada del método de Heap.

Se efectúa el segundo intercambio y se vuelve a marcar el elemento que va quedando en su lugar con color **azul cielo.**

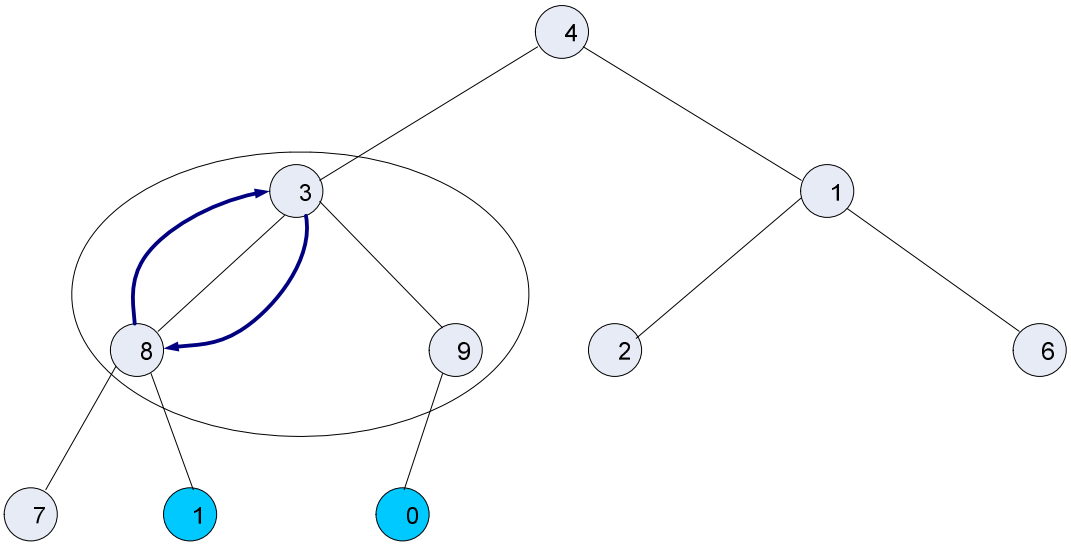
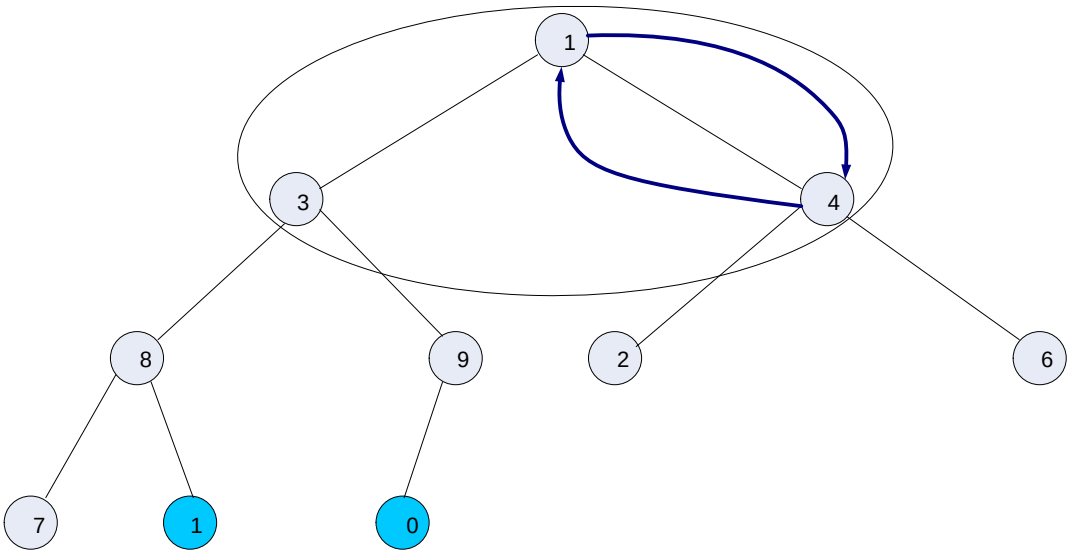


Figura 2.24. Análisis del tercer subárbol binario.

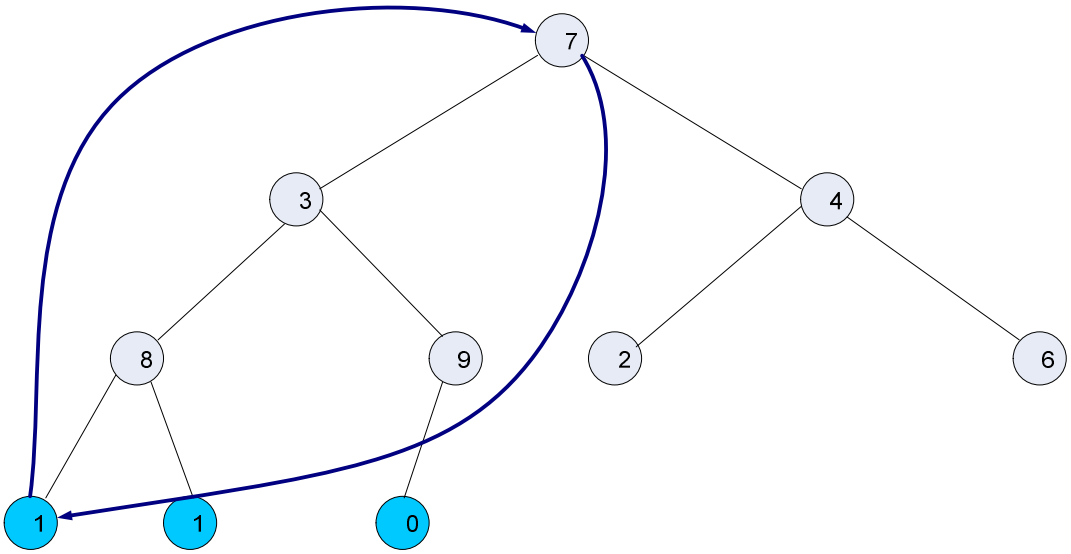
Se vuelven a analizar todos los subárboles del nivel más bajo y de derecha a izquierda, pero no se realiza ningún intercambio, ya que van cumpliendo con la estructura del **Heap**, hasta el tercero, el cual está encerrado en la elipse, en donde si se tuvo que realizar un intercambio del **3** por el **8**, para que cumpla con el **Heap**.



TERCER HEAP

Figura 2.25. Análisis del cuarto subárbol binario y tercer Heap.

En este ejemplo se realiza el intercambio del **1** por el **4** y se obtiene el tercer **H*eap***.



TERCER INTERCAMBIO

Figura 2.26. Fin de la tercera pasada del método de Heap.

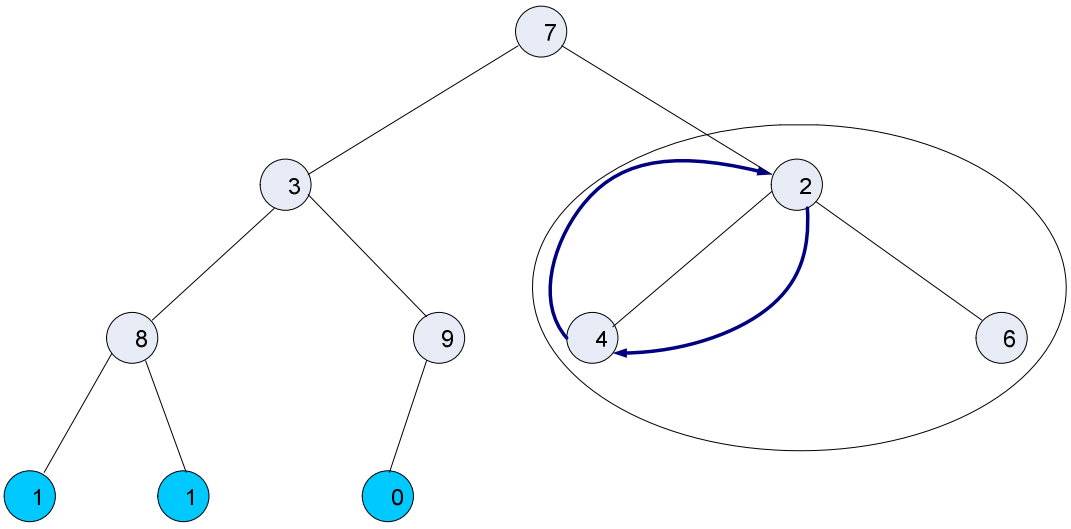
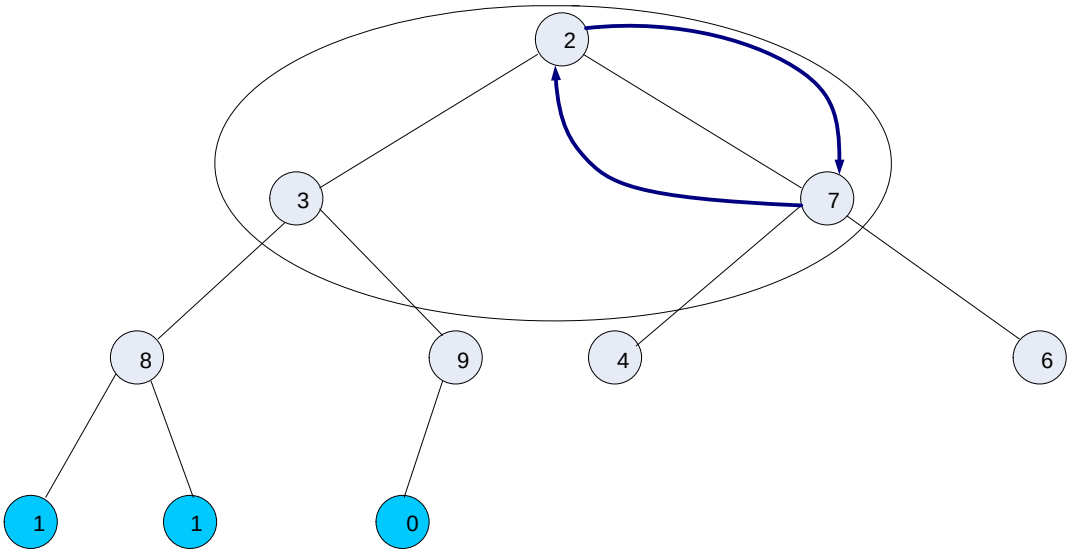


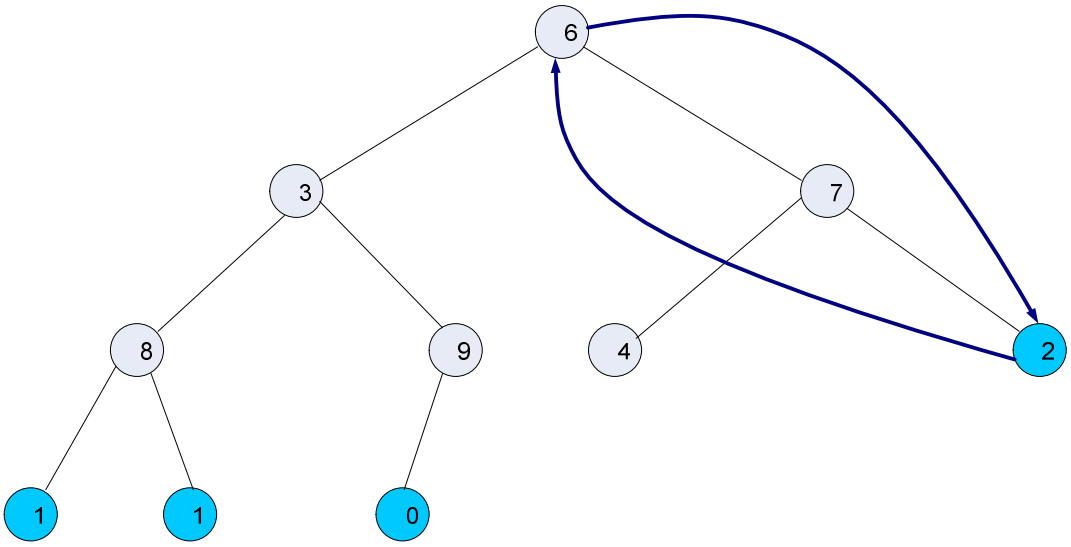
Figura 2.27. Análisis del primer subárbol binario.

En este ejemplo se lleva a cabo el intercambio del **2** por el **4** y el siguiente subárbol de ese nivel que se encuentra más a la izquierda (3, 8, 9), ya cumple con la estructura del ***Heap***, por lo que no se realiza ningún intercambio y se llega al subárbol de la raíz principal con el intercambio que a continuación se muestra.



CUARTO HEAP

Figura 2.28. Análisis del tercer subárbol binario y cuarto Heap.



CUARTO INTERCAMBIO

Figura 2.29. Fin de la cuarta pasada del método de Heap.

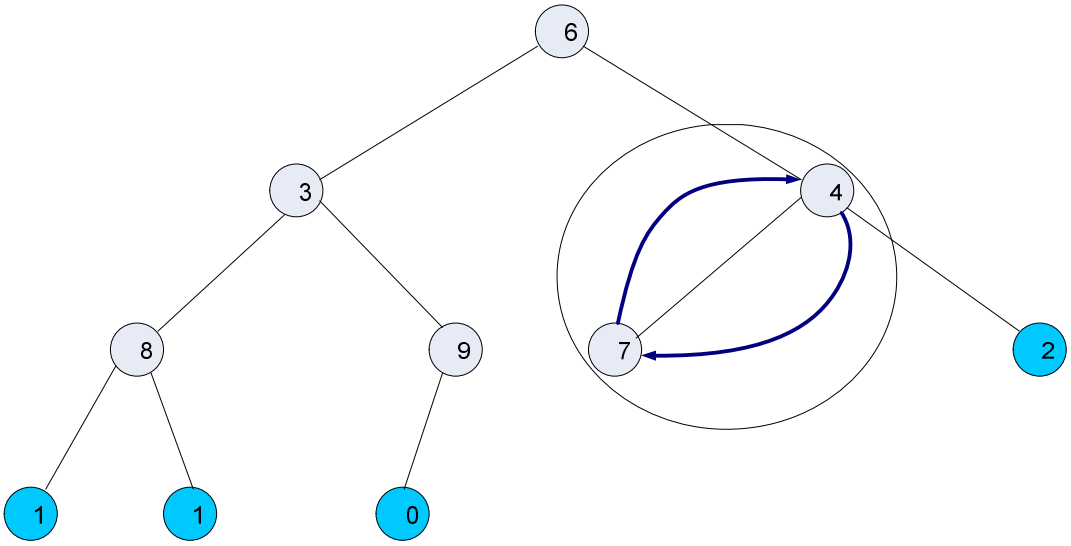
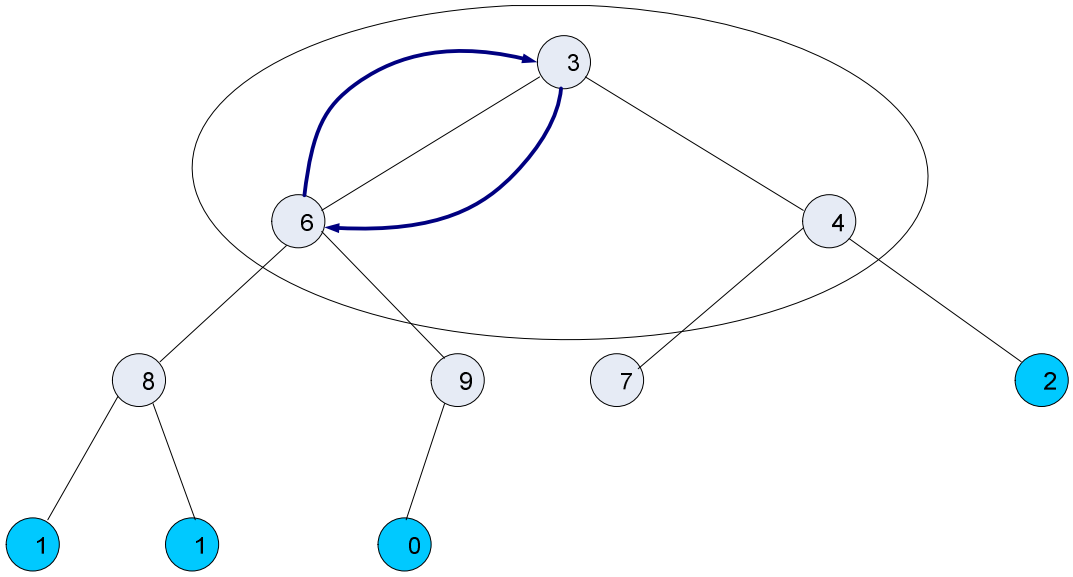


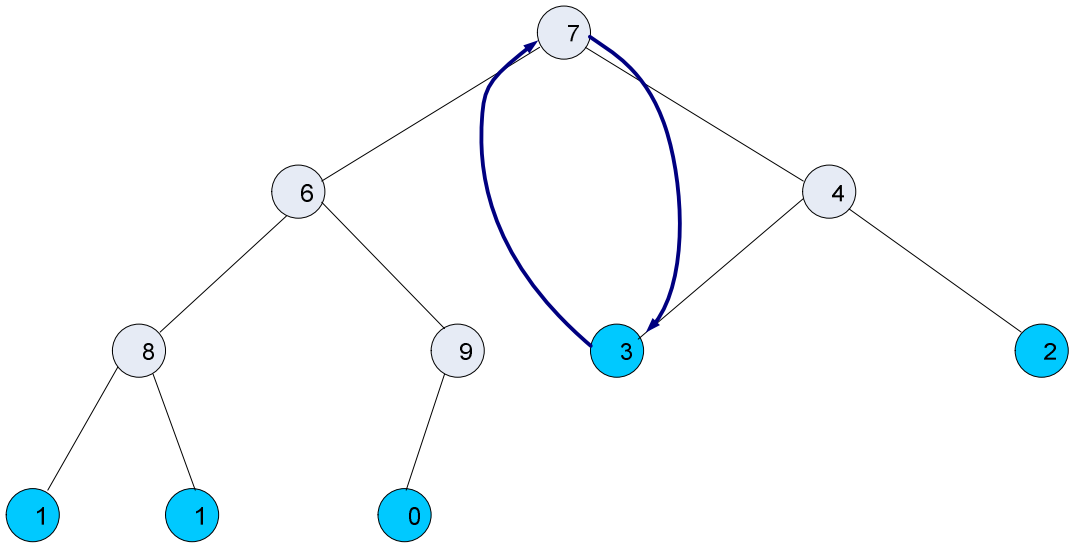
Figura 2.30. Análisis del primer subárbol binario.

Se intercambia el **4** por el **7** para ir formando la estructura del **H*eap***, al analizar lo que queda del árbol. El siguiente subárbol que queda de ese nivel (3,8,9), no sufre ningún intercambio, ya que sigue cumpliendo con la estructura del **H*eap***. Así que sólo queda la raíz principal para analizar e intercambiar los datos, que contienen esos nodos, como a continuación se muestra.



QUINTO HEAP

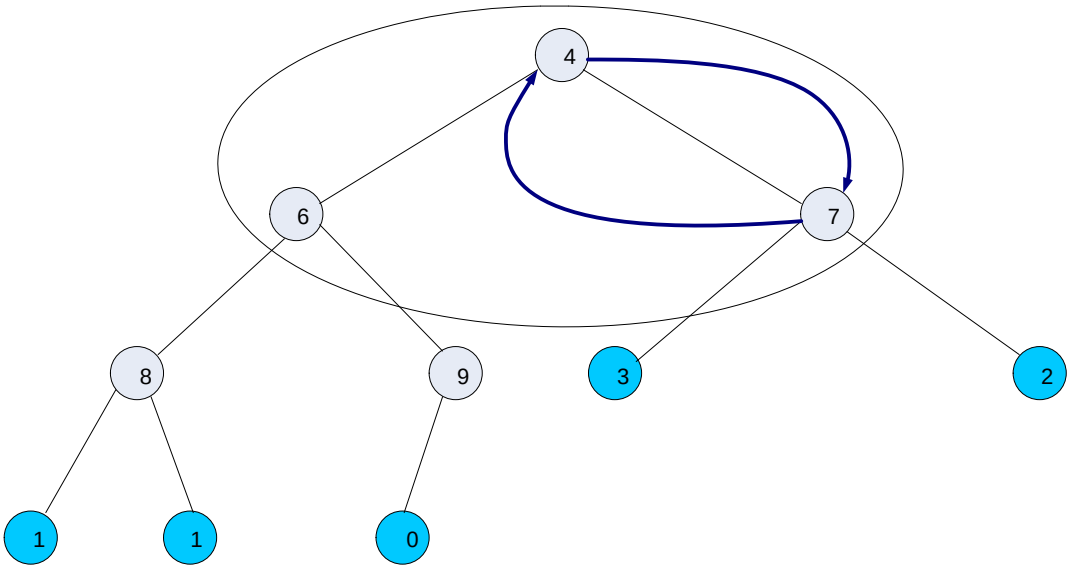
Figura 2.31. Análisis del tercer subárbol binario y quinto Heap.



QUINTO INTERCAMBIO

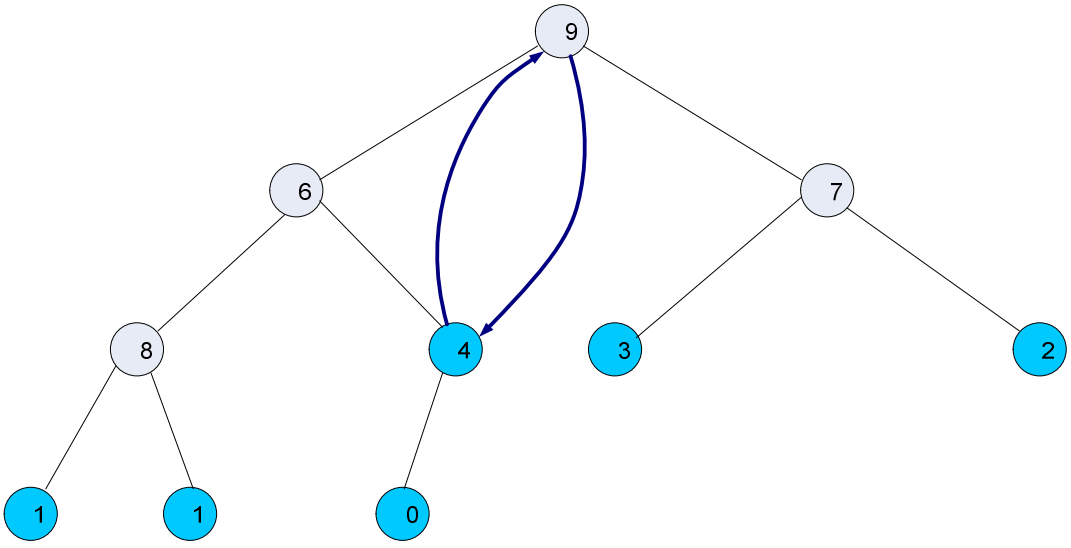
Figura 2.32. Fin de la quinta pasada del método de Heap.

El siguiente subárbol del nivel más abajo y más a la derecha (6,8,9) cumple con la estructura del ***Heap***. Sólo resta el de la raíz principal, donde se realiza el siguiente intercambio.



SEXTO HEAP

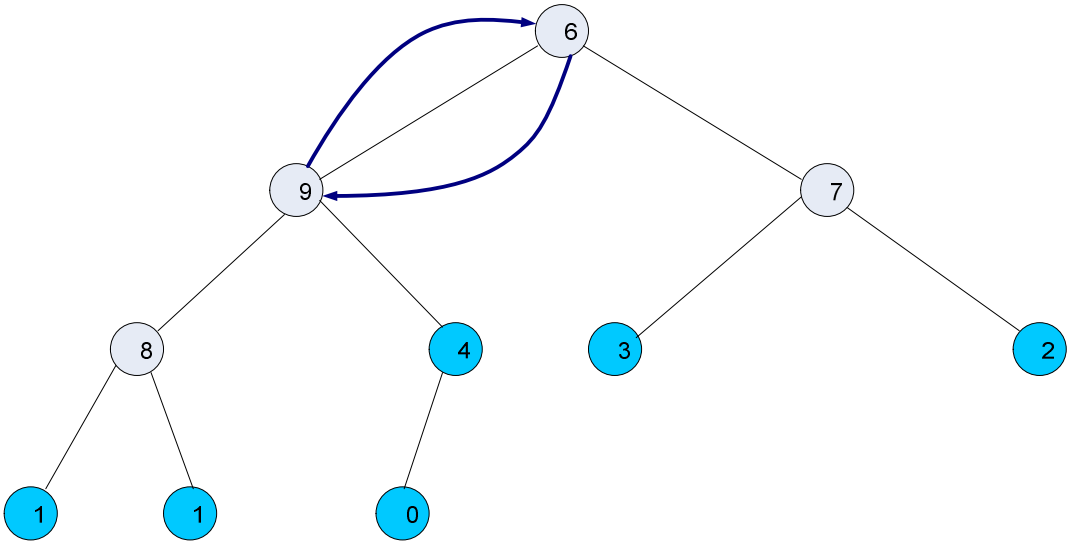
Figura 2.33. Análisis del segundo subárbol binario y sexto Heap.



SEXTO INTERCAMBIO

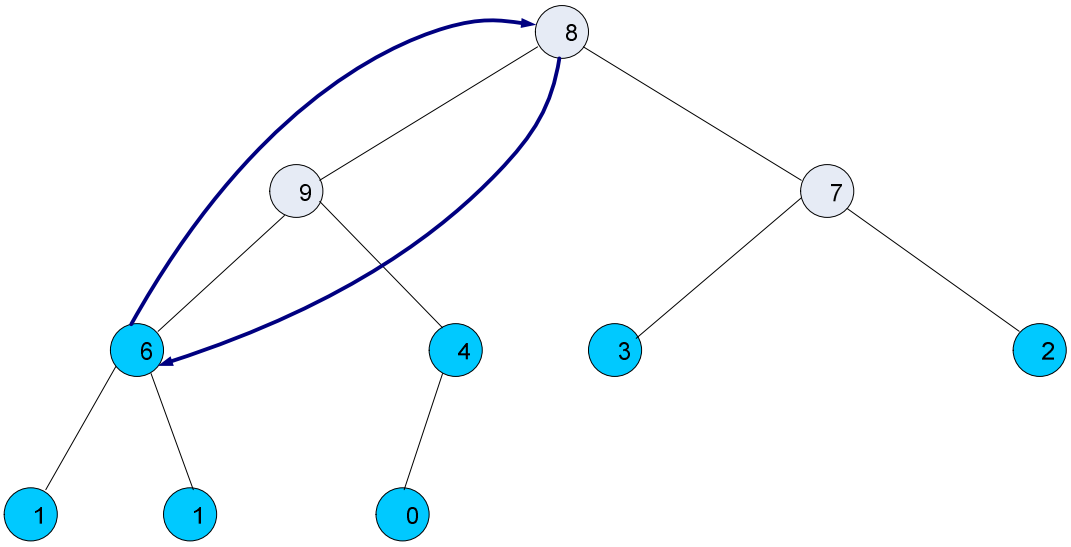
Figura 2.34. Fin de la sexta pasada del método de Heap.

El subárbol que se analiza para ir formando el siguiente **H*eap*,** ya no se modifica (6,8) y se prosigue a la raíz principal. Donde se realiza un intercambio, que se muestra enseguida.



SÉPTIMO HEAP

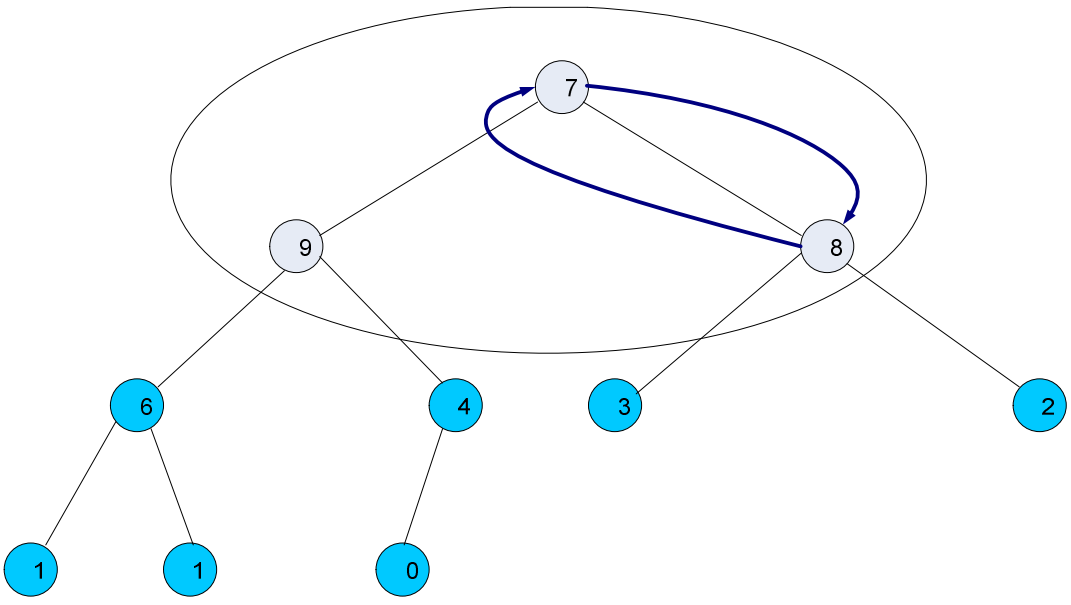
Figura 2.35. Análisis del segundo subárbol binario y séptimo Heap.



SÉPTIMO INTERCAMBIO

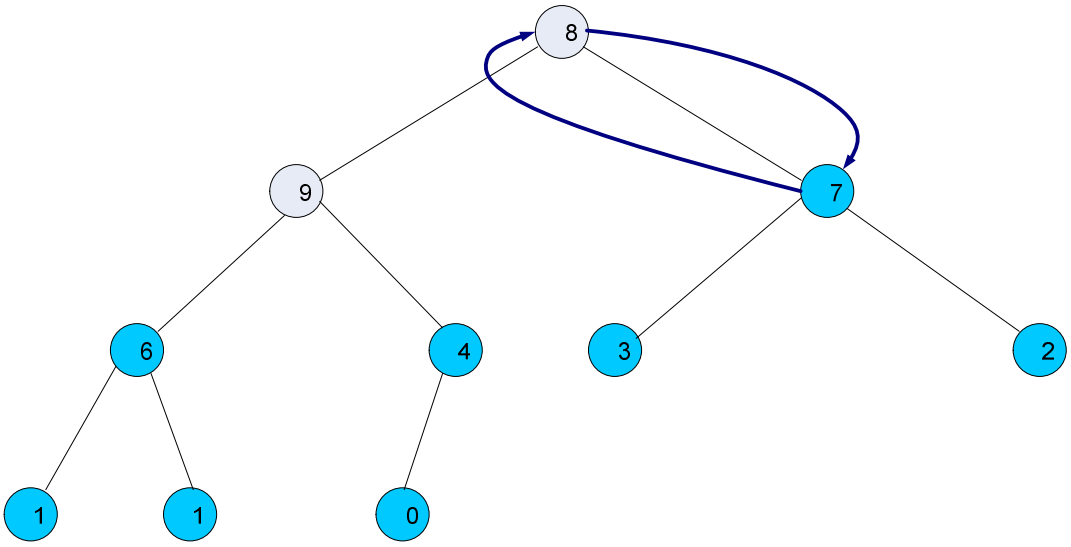
Figura 2.36. Fin de la séptima pasada del método de Heap.

Ya sólo queda analizar el subárbol de la raíz principal para formar la estructura del **H*eap***.



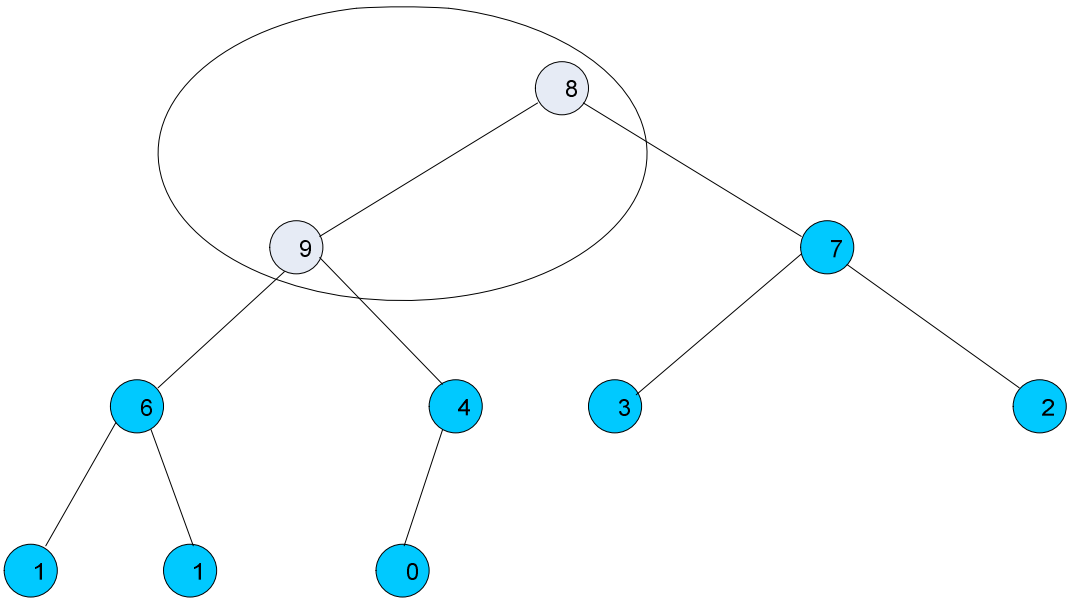
OCTAVO HEAP

Figura 2.37. Análisis del primer subárbol binario octavo Heap.



OCTAVO INTERCAMBIO

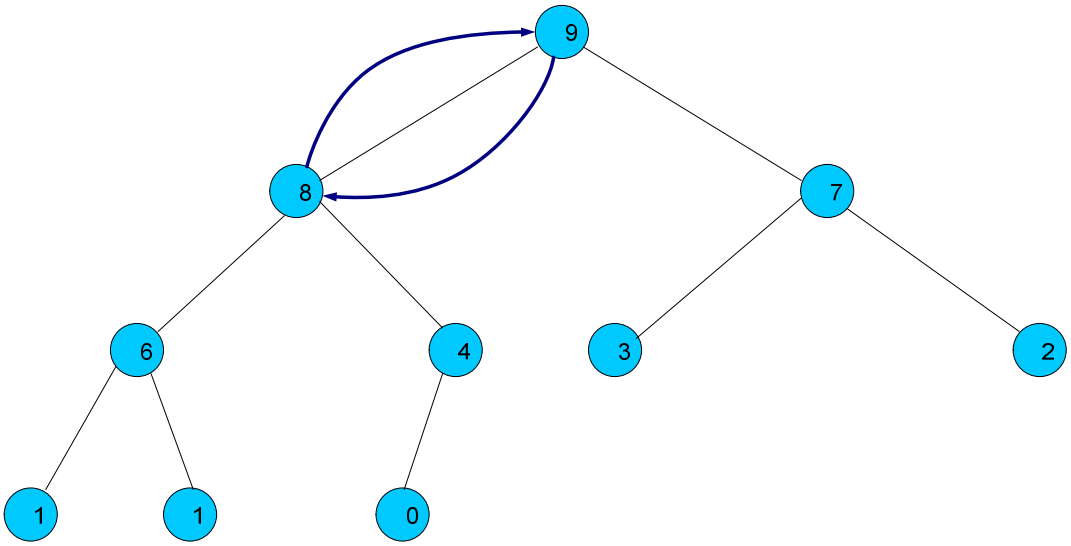
Figura 2.38. Fin de la octava pasada del método de Heap.



NOVENO HEAP

Figura 2.39. Análisis del primer subárbol binario noveno Heap.

Al comparar los valores de los dos nodos que quedan (8 y 9), no se realiza ningún intercambio, ya que como se observa cumple con la estructura de **Heap**.



NOVENO INTERCAMBIO

Figura 2.40. Fin de la novena pasada del método de Heap.

Si se observa, en cada pasada sólo se va acomodando un elemento, a excepción de la última, en la cual se colocan dos, parece a simple vista muy complejo y tardado el proceso, sin embargo, es uno de los métodos de clasificación más rápidos, ya que generalmente en la primera pasada, para encontrar el primer ***Heap***, es cuando se tarda en lograrlo, pero a partir de la segunda pasada, a pesar de que el resto de los elementos aparecen desordenados, los intercambios que se realizan, disminuyen considerablemente.

Analizar los subárboles es muy sencillo, ya que se trata de un árbol binario, el cual puede ser almacenado en un arreglo de una dimensión y, para recorrer sus ramas, basta con conocer la posición dentro del arreglo del nodo padre y multiplicarlo por dos, para que de la posición de la rama izquierda, y por dos más uno para encontrar la posición de la rama derecha.

De igual forma, para localizar la posición del nodo padre o raíz, basta con conocer la posición dentro del arreglo de la rama izquierda o derecha y dividirla entre dos, tomando la parte entera.

Para que funcione este recorrido en un arreglo de una dimensión y no existan problemas con el primer elemento, se sugiere almacenar estos elementos a partir de la posición **uno** del arreglo, ya que si se empieza en la posición **cero**, al multiplicar, siempre dará el valor de **cero**.

2.4. Método *shell*

Definición del método

La idea del método Shell es la de ir comparando de dos en dos los elementos, tomando una distancia entre uno y otro que se denominará **h**, la cual se empezará calculando aproximadamente de la mitad del número de elementos, siendo el primer elemento que se comparará con el que se encuentra aproximadamente a la mitad de todos los elementos; el segundo, se comparará con el que se encuentra aproximadamente con el que se encuentra a la mitad, más uno y así sucesivamente, mientras exista un elemento contra el cual comparar, en la primera pasada. Al número de elementos se manejará como **n**.

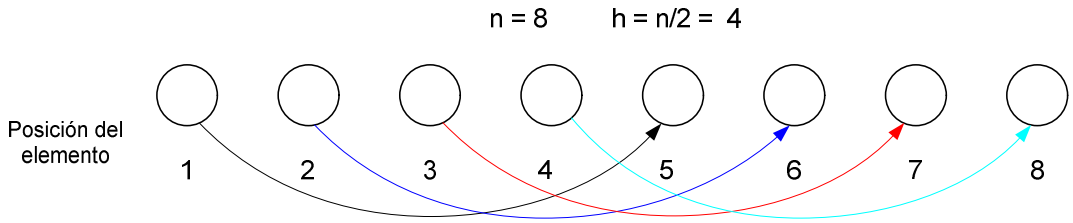
Se irán acomodando en la primera posición de los elementos que se comparan, el del valor mas pequeño si se trata de clasificación ascendente, o el mas grande, si es el caso de forma descendente.

En la segunda pasada, este incremento de **h** se reducirá a la mitad del valor que tenia en la pasada anterior, esto es, nuevamente se comparará el primer elemento con el que se encuentra aproximadamente a una cuarta parte de todos los elementos, el segundo con el que se encuentra a una cuarta parte más uno y así sucesivamente, hasta que exista algún elemento contra el cual comparar.

En cada una de las siguientes pasadas, el incremento se va reduciendo a la mitad de su valor, es decir, **h** se reducen a una cuarta parte, en la siguiente

pasada, a una octava parte de **n** y así en cada una de las pasadas, hasta que **h** llegue a ser **uno.**

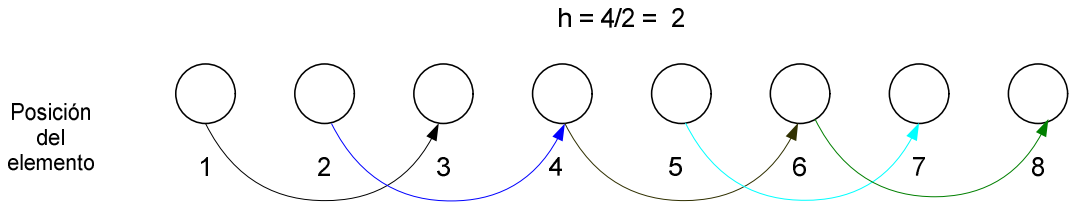
A continuación, se muestra en forma gráfica el manejo del tamaño de **h**, cuando el número de elementos a clasificar es de 8, con lo que le valor de **n**, se inicializa. Las comparaciones se darán cada 4 elementos, como se muestra con las flechas de colores, de tal forma que es el elemento que se encuentra en la posición 1,se comparará con el que se encuentra en posición 5, el que se encuentra en la 2 con el de la 6 y así sucesivamente.



shel01

Figura 2.41. Primera pasada del Shell.

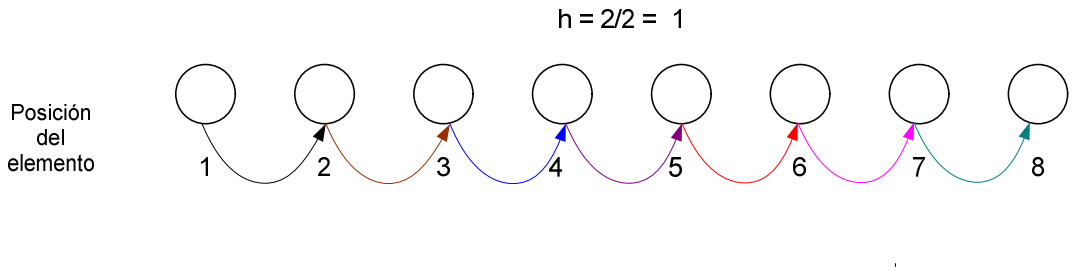
Ahora las comparaciones se realizarán cada 2 elementos, ya que **h**, se redujo a la mitad de su valor, en esta segunda pasada.



shel02

Figura 2.42. Segunda pasada del Shell.

Finalmente **h**, se vuelve a reducir a la mitad de su valor, quedado en 1.



shel03

Figura 2.43. Tercera pasada del Shell.

Es de resaltar que dependiendo de los incrementos de **h** y de la distribución de los datos que proporcione el usuario del sistema de información, puede o no clasificarlos correctamente, esto se ha visto en forma experimental, con varios conjuntos de elementos, donde se siguen todos los pasos del método. Este método debe su nombre a su creador D.L. Shell quien lo dio a conocer en 1959.

Para el pseudocódigo, que a continuación se presenta, se van a manejar los incrementos de **h** en un arreglo llamado **incr**. El número de elementos del arreglo **incr** va a estar dado por las variables denominadas **ele** y en tanto que **datos** es el arreglo donde estarán los datos tanto desordenados como ya clasificados al final del algoritmo.

Es importante indicar que el algoritmo que viene descrito a continuación, funciona cuando se empieza con una **h** igual a **n/2**, ya sea la parte entera o redondeada, ya que si se toma por ejemplo un valor menor a esa parte entera, el algoritmo falla.

El pseudocódigo del algoritmo es el siguiente:

```
inicio
        indice = 0
        mientras indice  < ele
                h = incr [indice]
```

```
                j = h
                mientras j < n
                        i = j – h
                        k = datos [ j ]
                        repetir
                                si k < datos [ i ]      // con < es ascendente y
                                                        //          con > es desc.
                                        datos [i + h] = datos [ i ]
                                        i = i – h
                                en caso contrario
                                        break    // rompe el ciclo de repetir
                                fin
                        hasta i <= 0
                        datos [ i + h ] = k          // se realiza el intercambio, en
caso que se de
                        j = j + 1
                        fin
        indice = indice + 1
        fin
fin
```

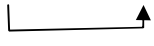
Ejemplo

El ejemplo que a continuación se presenta, se requiere en forma ascendente y utiliza los siguientes datos:

El número de elementos **n** es igual a 6, por lo que el primer incremento **h** es igual a 3 (**n/2 o sea 6/2**)

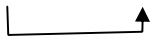
Con h = 3

1, 3, 4, 0, 9, 3



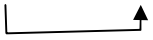
Se realiza un intercambio **1** por **0** y se comparan los dos datos que siguen:

0, 3, 4, 1, 9, 3



No se realiza ningún intercambio.

0, 3, 4, 1, 9, 3



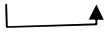
Se realiza otro intercambio: **4** por **3.**

0, 3, 3, 1, 9, 4

Fin de la primera pasada.

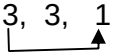
Con h = 2

0, 3, 3, 1, 9, 4



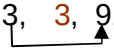
No hay intercambio y se sigue la comparación de los otros dos datos

0, 3, 3, 1, 9, 4



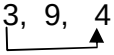
Hay un intercambio de **3** por **1**

0, 1, 3, 3, 9, 4



No hay intercambio.

0, 1, 3, 3, 9, 4



No hay intercambio.

Fin de la segunda pasada.

Con h = 2

Con h = 1

0, 1, 3, 3, 9, 4

Sin intercambio

0, 1, 3, 3, 9, 4

Sin intercambio

0, 1, 3, 3, 9, 4

Sin intercambio

0, 1, 3, 3, 9, 4

Sin intercambio

0, 1, 3, 3, 9, 4

Se intercambia el **9** por el **4**

0, 1, 3, 3, 4, 9

Fin de la tercera pasada.

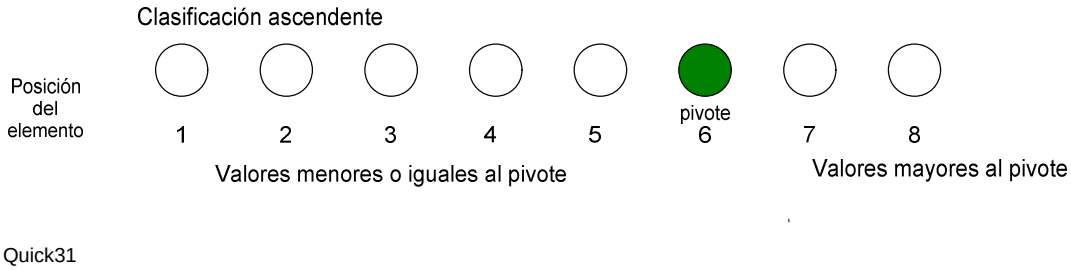
Quedan clasificados los datos.

2.5. Ordenación rápida (Quick Sort)

Definición del método

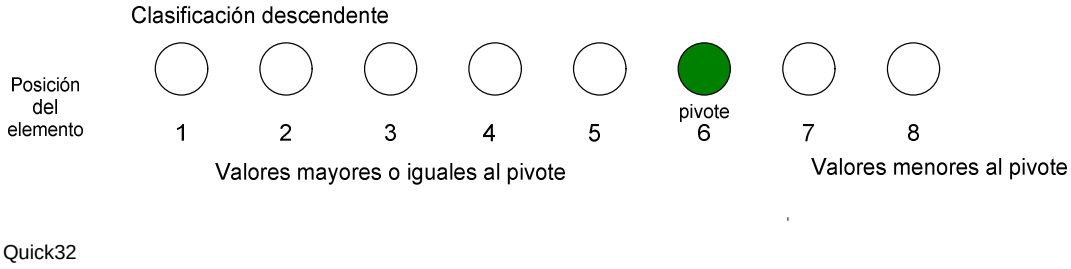
El algoritmo de ordenación rápida **(Quick Sort)** consiste en tomar el primer elemento de la lista original de datos, al cual se le llama **pivote** y acomodarlo en otra posición dentro de la lista, de tal forma que ya quede en una posición, de acuerdo a su valor, por ejemplo, si se está clasificando en forma ascendente todos los elementos de la lista, menores o iguales a este **pivote,** se colocarán

del lado izquierdo y los mayores del lado derecho, si se trata de una clasificación en forma descendente sería al revés, todos los elementos con valor mayor o igual del lado derecho y los menores del lado izquierdo.



Quick31

Figura 2.44. Pivote del Quick en forma ascendente.



Quick32

Figura 2.45. Pivote del Quick en forma descendente.

Una vez colocado el pivote en la posición que le corresponde, es posible que la lista quede dividida en dos partes, si es que el valor del pivote está en un valor intermedio con respecto a los valores del resto de los elementos y quedaría en un lugar también intermedio de la lista, pero puede ser que si el valor del pivote fuese el más chico de todos, entonces el pivote quedará al principio de la lista, y si fuera el mayor de todos los elementos, quedará al final de la lista y, en ambos casos, la lista no se dividiría.

Estos pasos para colocar el pivote en el lugar que le corresponda se repite para cada una de las sublistas, que aparecen después de dividir la lista original. Cada pivote que se vaya colocando se puede ir marcando de alguna forma

para ya no tomar en cuenta ese elemento, o bien las sublistas serán las que se marquen, indicando que en ellas es donde se seguirá aplicando el método.

Donald Knuth, sugiere el uso de una Pila, para dividir la lista, donde cada uno de sus elementos, es el par de valores de inicio y fin de cada una de las sublistas.[10]

El método es muy rápido, como su nombre lo indica **Q*uick***, ya que se aplica el mismo algoritmo para cada sublista y además se pude utilizar recursivamente o por auto-referencia para cada una de las sublistas.

Lo que pareciera difícil del método es saber en qué posición dentro de la lista se colocará el pivote**,** ya que depende de los valores del resto de la lista, por lo que es necesario analizar todos los demás elementos antes de colocar dicho pivote. Pero es bastante sencillo este paso como a continuación se describe:

Se van a tener en un arreglo o archivo todos los elementos a clasificar y se requerirán de tres variables de tipo apuntador, una llamada **p** (por ejemplo), que indica el lugar del **pivote**, que originalmente esta en la primera posición, otra **i,** que empieza en la segunda posición y se va a mover hacia la derecha mientras los valores de los elementos a los que esta apuntando sean menores o iguales al del **pivote**, si se clasifica en forma ascendente o mayores o iguales si es en forma descendente.
Existe otra variable que se denominará **d**, la cual estará inicializada en la última posición de la lista y se moverá hacia la izquierda, en tanto que los valores a los que apunta sean mayores al **pivote**, si se clasifica en forma ascendente o menores si es en forma descendente.

---

[10] **Knuth, Donald E.**, *El arte de programar ordenadores, Volumen III Clasificación y Búsqueda*, (impresión.), ESPAÑA, REVERTÉ, 1987, pag 122-125

La variable **i** se inicializará en la posición del segundo elemento de la lista o sublista de datos a clasificar, con el fin de comparar el dato de esa posición con el valor del **pivote**, que es donde está apuntando **p** (p, es la dirección dentro del arreglo, y pivote es el valor del elemento que está siendo apuntado por p).

El apuntador **i** avanza a la derecha, incrementándose de uno en uno, mientras los valores a los cuales va apuntando sean menores o iguales al valor de donde esta apuntando p (pivote), si se trata de una **clasificación ascendente** y sería al revés si se trata de forma **descendente**.

En el momento en que **i** apunte a un elemento que sea mayor al valor de donde apunta **p** (forma ascendente), entonces se detiene, esperando hacer un intercambio.

En tanto que el apuntador **d** avanza a la izquierda, decrementando su valor de uno en uno, mientras que los valores a los que va apuntando sean mayores (forma ascendente) al valor donde esté apuntando **p**.

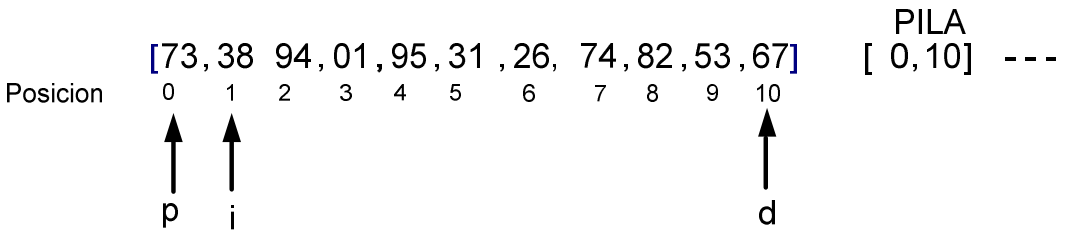
Cuando **d** apunte a un elemento que sea menor al que apunta **p**, se detiene y es cuando se realiza el intercambio por el elemento que está apuntado por **i**. Posteriormente ambos apuntadores **i** y **d**, siguen avanzando cada uno por su ruta y así se van hasta que se vuelvan a detener cuando se cumplan las condiciones anteriormente mencionadas, nuevamente se realizará el intercambio de elementos hasta que se crucen los apuntadores **i** y **d**, estos es cuando la posición a la que apunta **i** sea mayor a **d**, es cuando se podrá colocar el elemento apuntado por **p**, en la posición que le corresponde, ya que se intercambiará el elemento apuntado por **d**, por el que es apuntado por **p**.

Con el ejemplo que se muestra a continuación, se esclarecerán estos pasos, en el cual la lista y sublistas que se generarán, se irán encerrando entre corchetes **[ ]**, cuando quede un sólo elemento en la sublista ya no se marcará

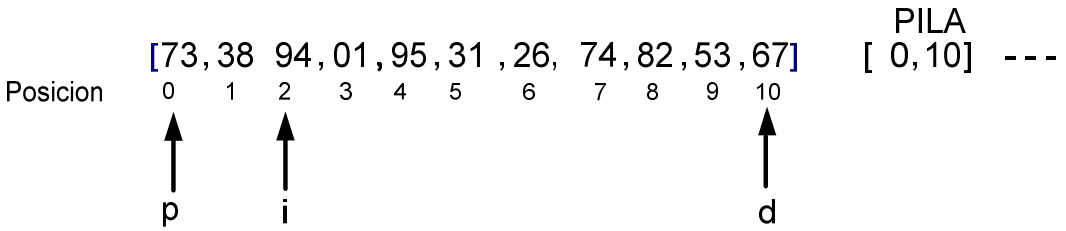
dicha sublista. También es importante resaltar que los elementos de la lista se encuentran almacenados a partir de la dirección **cero**.

Ejemplo

A continuación se presenta un ejemplo, el cual se clasificará en forma ascendente con los siguientes datos:

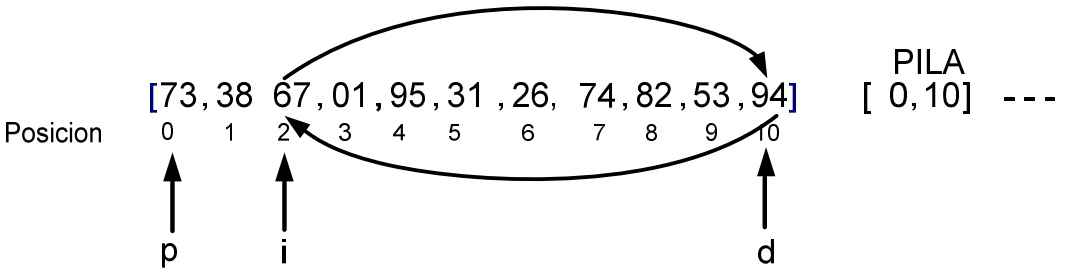


Se inicializan los apuntadores de **p** en el primer elemento, **i** en el segundo y **d** en el último. En tanto que la pila tiene dos valores que indican las posiciones, a través de las cuales se consideraran la lista a analizar, en este caso es a partir del elemento que se encuentra en la posición **cero**, que es la primera de la lista, al elemento que esta en la posición **diez**, siendo la última de dicha lista.

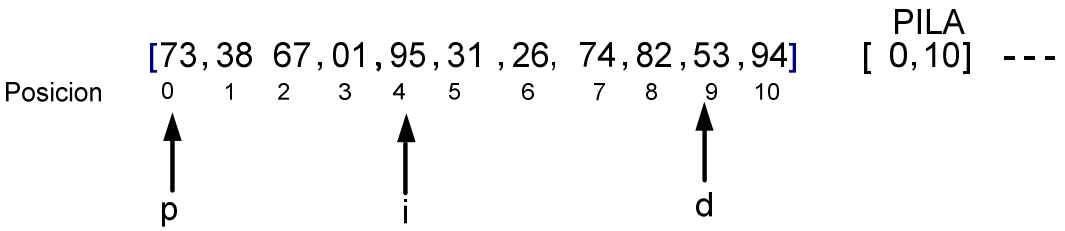


i avanza a la derecha (incrementándose en uno) mientras esté apuntando a un valor menor o igual al que apunta **p**, por lo que avanzó, hasta que apunta al 94, ya que es un valor mayor a 73, en tanto **d**, avanza hacia la izquierda (decrementandose en uno) mientras el valor al cual apunte sea mayor en este

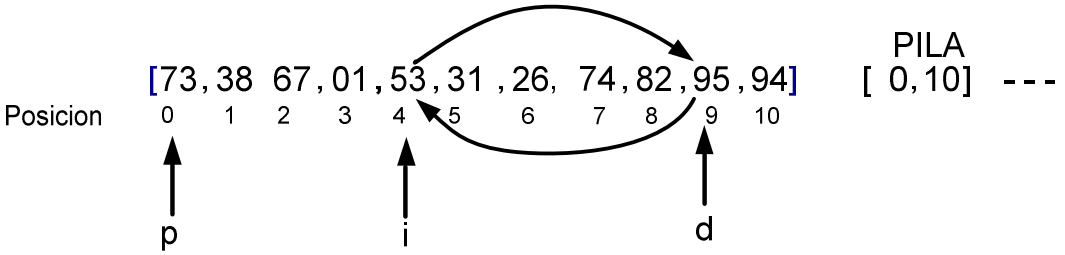
caso a 73 (valor del pivote apuntado por p) y se detiene en 67, porque ya no cumple con esa condición.



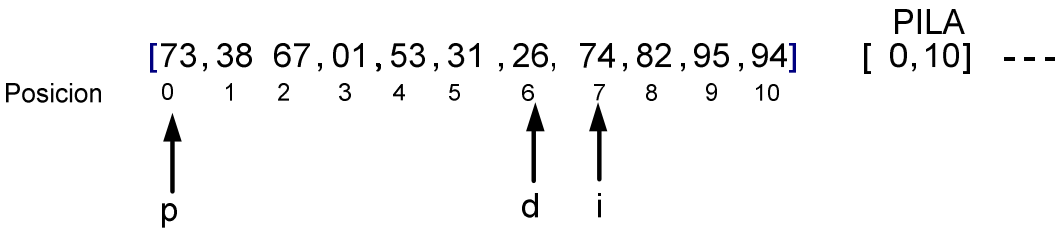
Se realiza el intercambio del elemento que está apuntando **d** por el que está apuntando **i**.



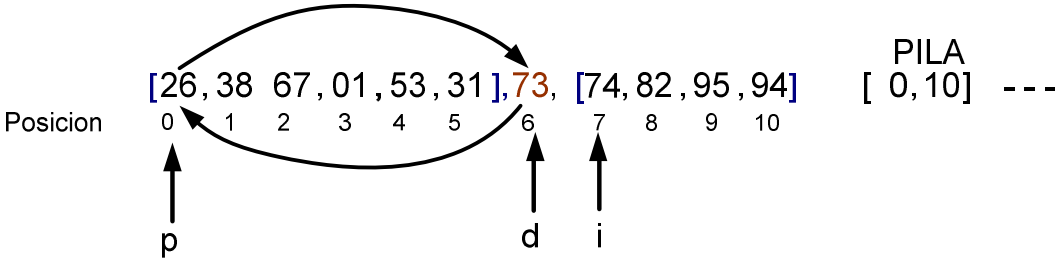
El apuntador **i** se detiene en 95 ya que es mayor a 73 y **d** se detiene en 53, porque es menor a 73.



Se intercambia el 53 por el 95 y seguirán avanzando los apuntadores.

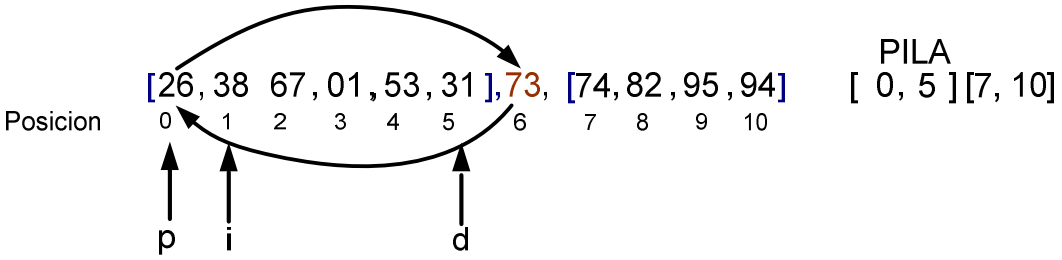


En este momento se traslapan los apuntadores **i** y **d**, siendo **i** mayor a **d**, con lo cual ya se sabe en qué posición debe quedar el **pivote**.



Se acomoda en su posición correcta el **pivote**, que es el elemento que está apuntado por **p**, a la posición donde está apuntando **d** y las lista original, se subdivide en dos, cada una con sus corchetes [ ], y el número que se acomoda, queda marcado con color marrón. De la pila se retira el único elemento que tiene, pero ahora se ingresan dos ya que se subdividió la lista en dos.

Con este paso termina la primera pasada.

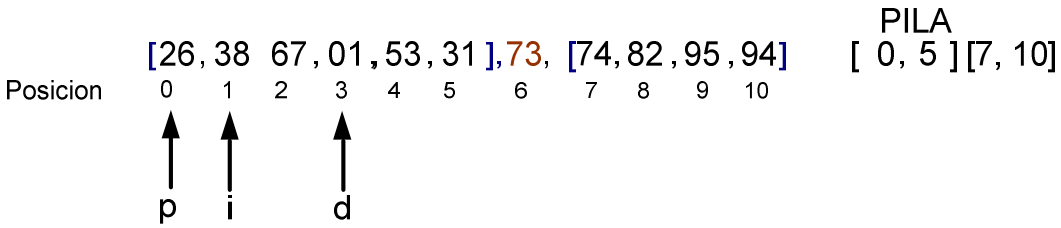


Ahora la pila tiene dos elementos, ya que la lista se dividió en dos sublistas, siendo, la primera de ellas los elementos que se encuentran de la posición 0

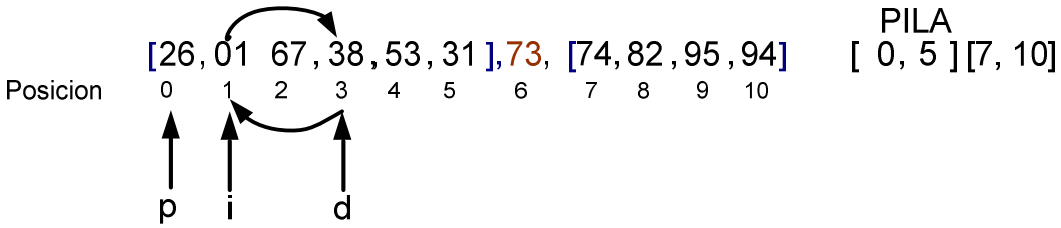
(cero) a la 5 y la segunda de ellas los elementos que se encuentra de la posición 7 a la 10.

Fin de la primera pasada.

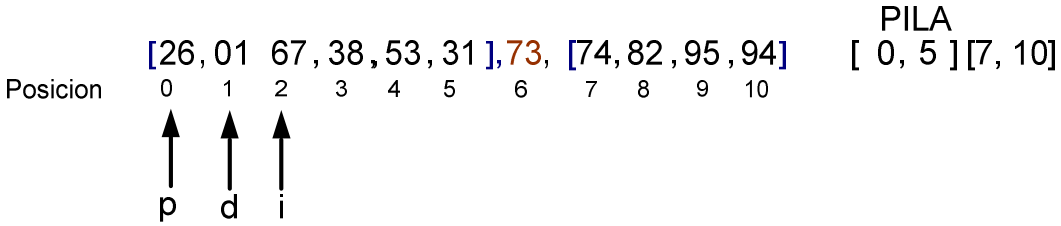
Se vuelven a inicializar los tres apuntadores (**p, i** y **d**) en la primera de las sublistas, como se realizó en el inicio del ejemplo y se agregan a la pila dos elementos, uno para cada sublista, donde vienen los limites en cuanto a las posiciones donde se encuentran éstas.



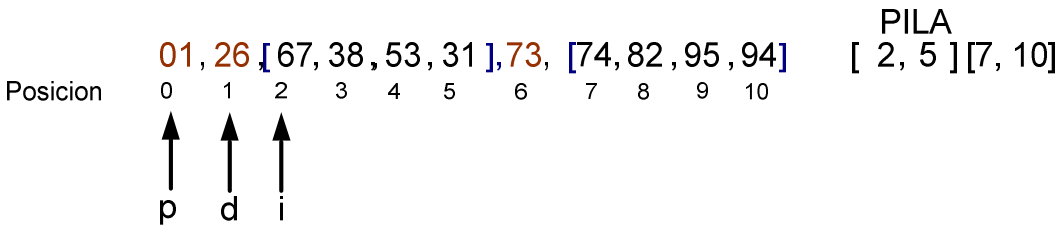
Los apuntadores se detienen cuando ya no se cumplen las condiciones de comparación.



Se realiza el intercambio de elementos



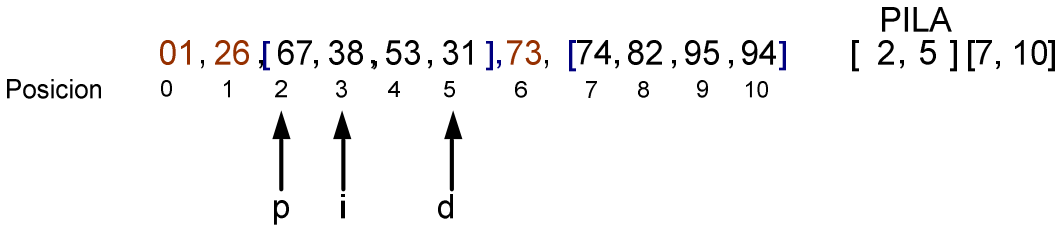
Se traslapan nuevamente los apuntadores **d** y **i**, estos es **i** > **d** listo para poner el pivote en la posición que le corresponde.



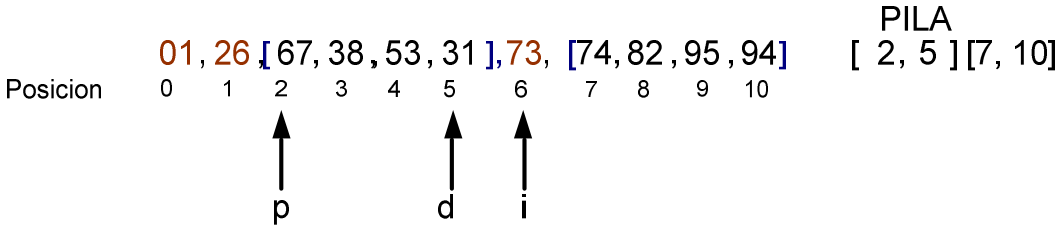
Se realiza el intercambio del elemento apuntado por **p** y por el de **d**, marcando el **pivote** que es el 26 por quedar en la posición que le corresponde con color **marrón**, en esta ocasión, también se marca el 01, ya que quedaría en una sublista de un solo elemento, ya que si existe un solo elemento en una sublista esta se encuentra ya clasificada, por no tener otro elemento contra quien comparar.

También se recorre el corchete izquierdo, ya que ahora la sublista queda a partir de la posición 2.

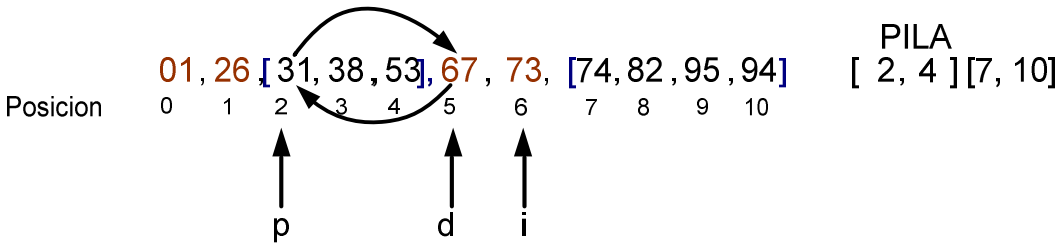
Fin de la segunda pasada.



Se inicializan nuevamente los tres apuntadores, como si fuera la primera vez, pero ahora en la nueva sublista y se procede con el método.

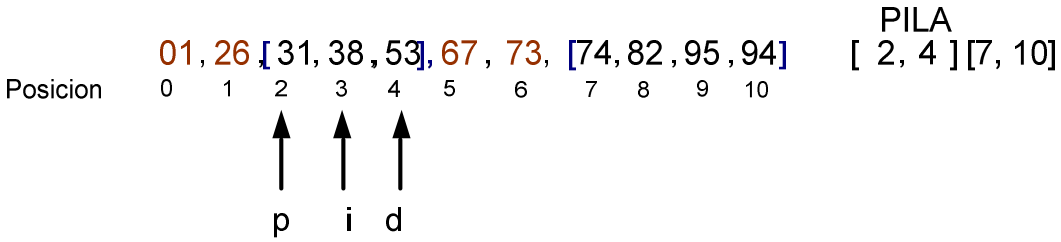


Avanzan en sus respectivas rutas los apuntadores **i y d,** al no poder seguir se traslapan **(i > d)**.

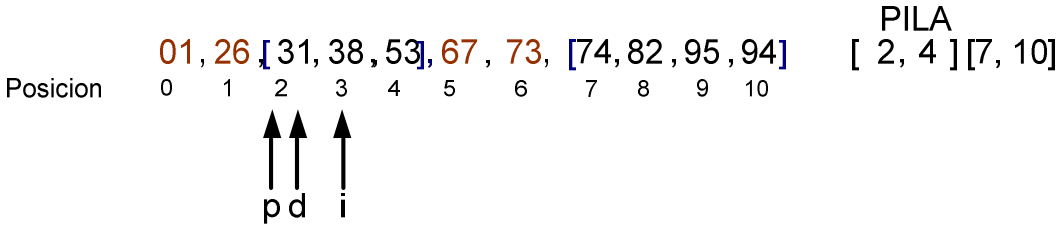


Se acomoda el **pivote**, se marca con color **marrón** y el elemento de la pila, se actualiza con los nuevos valores de los límites de la nueva sublistas (ahora 2 y 4).

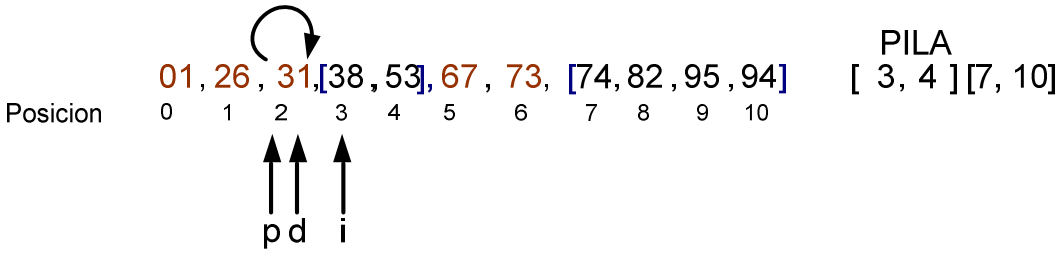
Fin de la tercera pasada.



Se inicializan los apuntadores.

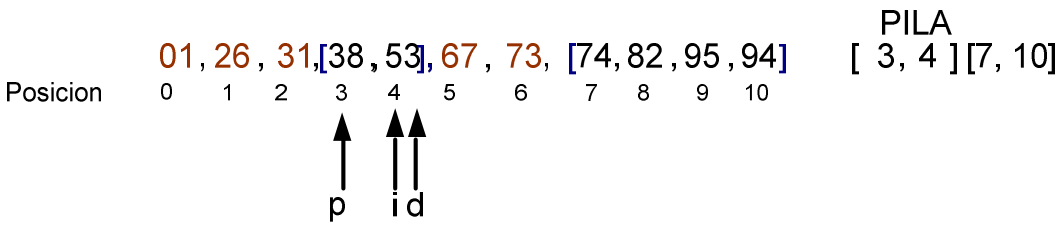


Traslape de apuntadores el valor de la posición **i > d**

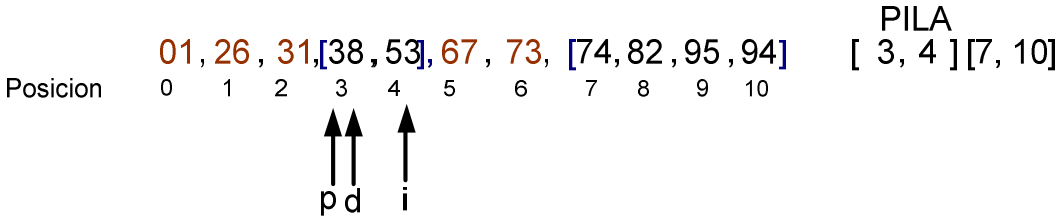


Se realizaría el intercambio de lo que apunta **p** por lo que apunta **d**, pero como están apuntando a la misma posición, no se hace ningún intercambio. Solamente se marca y se actualiza el límite de la sublista.

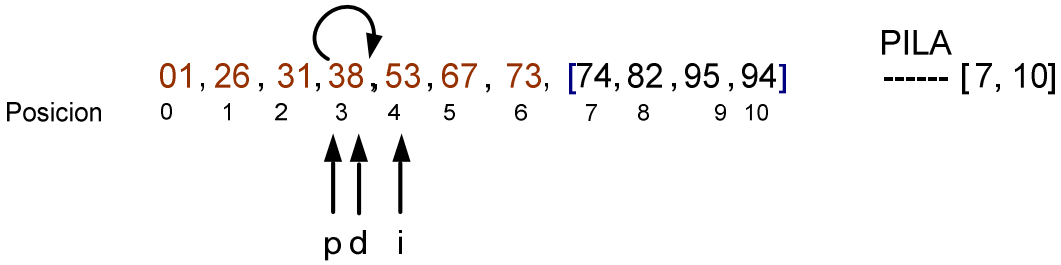
Fin de la cuarta pasada.



Se inicializan apuntadores.

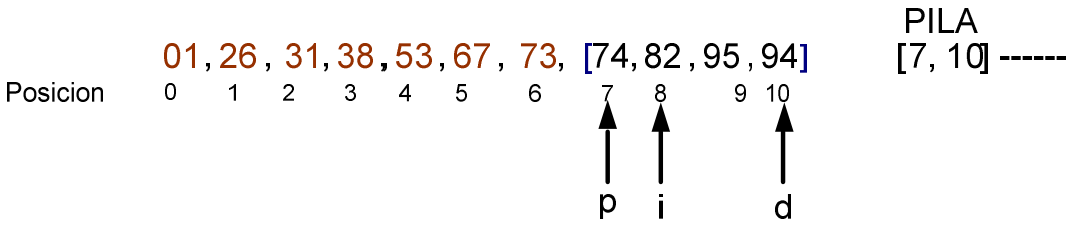


Se traslapan apuntadores.

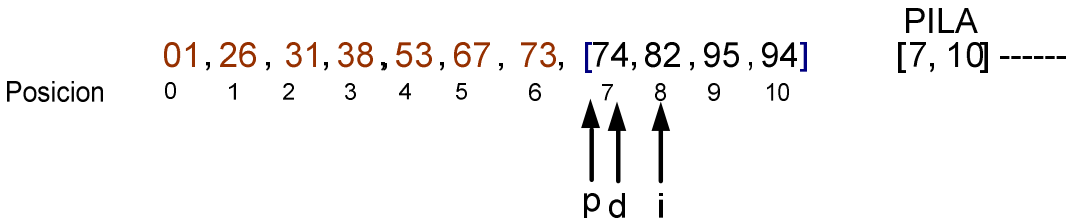


No se realiza ningún intercambio, ya que **p** y **d** apuntan a al misma dirección y el 38 se marca aparece con color **marrón** como el **pivote**, pero también el 53, debido a que sólo queda él en esa sublista, también se actualizan los elementos de la pila, con los guiones - - -, lo cual significa que ese elemento ya salió de dicha pila.

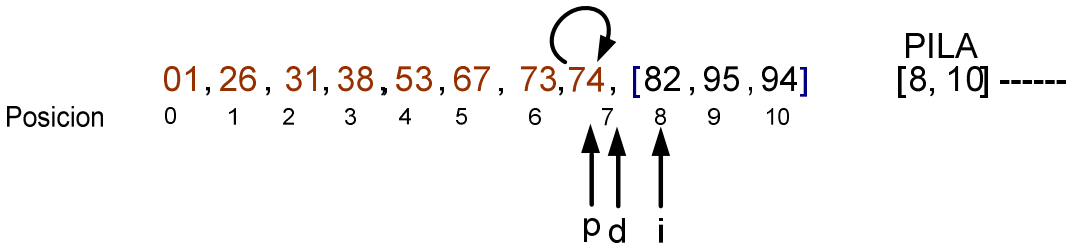
Fin de la quinta pasada.



Inicialización de los apuntadores.

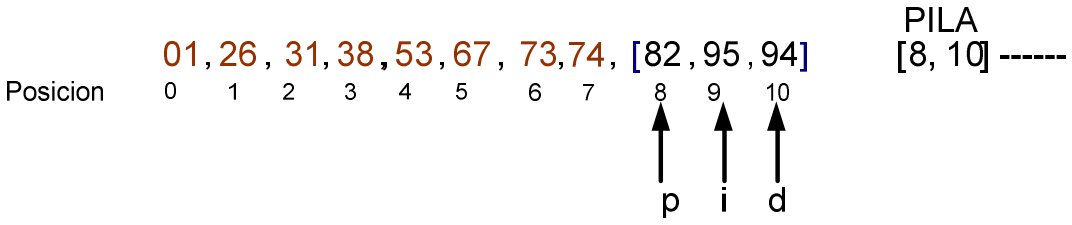


Se mueven los apuntadores **i > d**

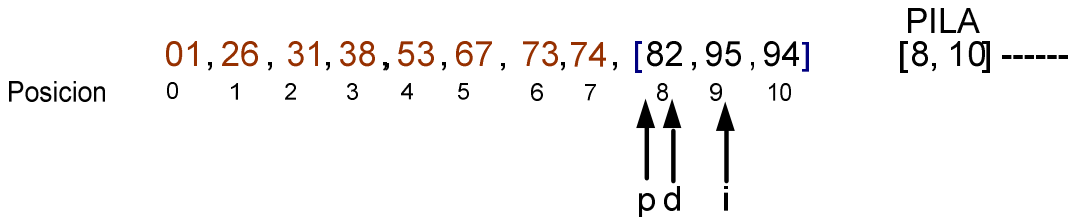


No se realiza ningún intercambio, ya que **p** y **d** apuntan a la misma posición, sólo se actualiza el límite de la sublista.

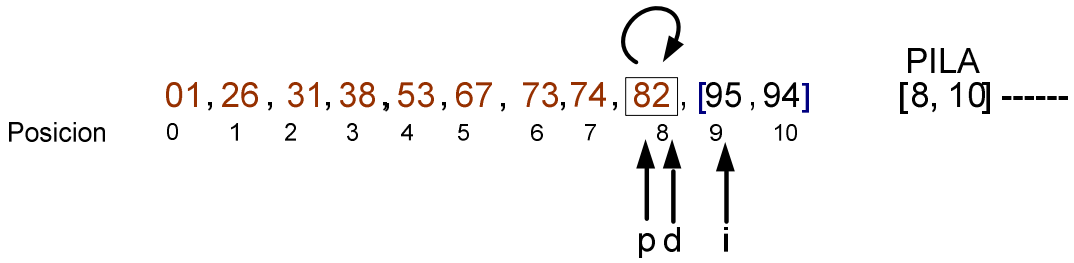
Fin de la sexta pasada.



Apuntadores inicializados.

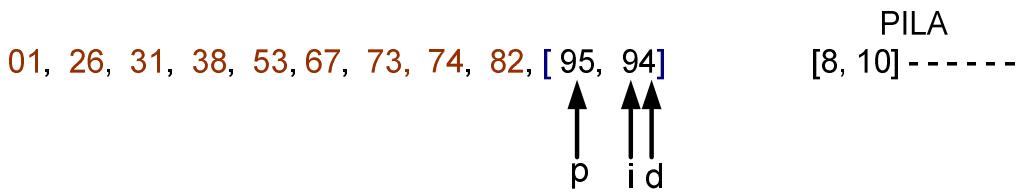


i > d

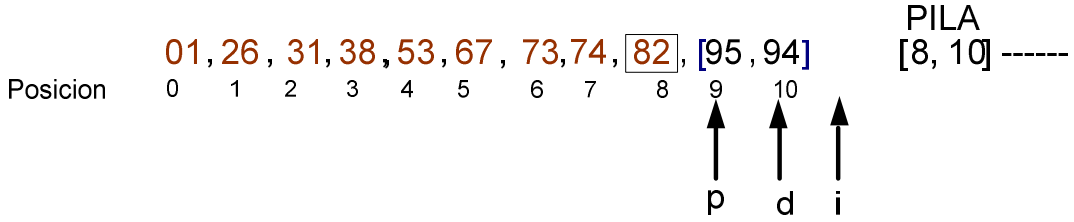


No se realiza ningún intercambio. **p** y **d** apuntan al mismo elemento, se actualiza el limite de la sublista que queda.

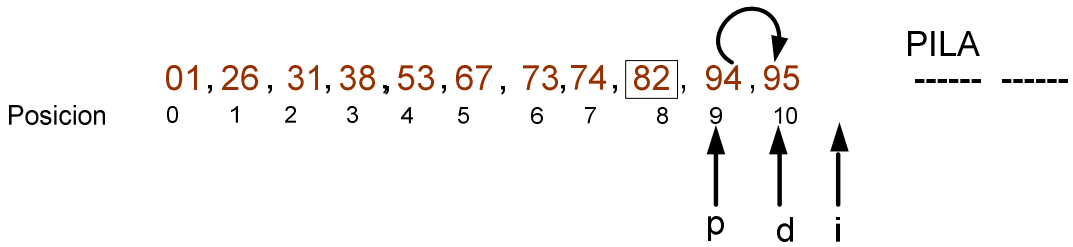
Fin de la séptima pasada.



Apuntadores inicializados.



Se vuelven a traslapar los apuntadores **i** y **d.**



Se realiza el intercambio de lo que apunta **p** por lo que apunta **d**, quedando marcado con color **marrón**, el pivote, que es el valor de 95 y como el 94 es un solo elemento, también queda marcado. La pila queda vacía y termina la ejecución de este método.

Los elementos quedan clasificados en forma ascendente.

Fin de la octava pasado.

Este método, parece complicado, pero en realidad, es uno de los más rápidos, en cuanto a su ejecución, no es tan difícil programarlo.

Algo importante de señalar, es que se pueden tener varios campos o llaves para clasificar, en los ejercicios mostrados solo se tomó uno solo, pero por ejemplo para poder generar las listas de cada grupo de clases de una facultad, es necesario realizar las siguientes clasificaciones y en el orden que se van mencionando:

Primero por carrera, ya que una facultad, puede tener más de una carrera y cuando se inscriben los alumnos, todas las solicitudes, pueden estar desordenadas, si se trata de un solo archivo de inscripciones. De esta forma se tendrán juntos todos los registros (solicitudes) de los alumnos de cada carrera, después de clasificar por este campo.

Después por materia, para agrupar todos los alumnos que son de una determinada carrera y de una determinada materia.

A continuación se tomaría el campo de grupo.

Finalmente el del nombre, para que al final de estas cuatro clasificaciones, se tuvieran listas de grupos (que son las que se les dan a los profesores) donde vendrían todos los alumnos de una determinada carrera, materia, grupo y además en orden alfabético (puede ser en forma ascendente o descendente).

2.6. Criterios de selección del método de ordenamiento

Elegir un algoritmo, cuando se tienen dos o más algoritmos para solucionar el mismo problema, se tienen que tomar en cuenta algunos criterios, como los que a continuación se mencionan::

"Que el algoritmo sea fácil de entender, codificar y depurar

Que el algoritmo use eficientemente los recursos del computador y, en especial, que se ejecute con la mayor rapidez posible"[11]

En lo que se refiere al primer punto, se contemplan las siguientes características:

- Cuando se ocupe el algoritmo, pocas veces, esto es importante, ya que el costo de tiempo de programación es mucho mayor al costo de ejecución del programa, pero si resulta que el programa se va a utilizar muchas veces, entonces el costo de ejecución del programa, puede llegar a ser mucho mayor que el tiempo de la escritura del programa, más si el tamaño de la entrada pueda llegar a ser muy grande.
- Resulta más económico tener un algoritmo que sea más eficiente (más complejo), siempre y cuando se vea reflejado en un tiempo de ejecución del programa mucho menor, a que si se trata de un programa donde se utilice un algoritmo más sencillo. Para poder determinar la variación de tiempos, pues sería necesario el implantar varios algoritmos, para comparar tanto la complejidad, como sus tiempos. Por lo cual los desarrolladores de sistemas de información, deben de contemplar cuando realizar un algoritmo

---

[9] Alfred V. Aho, *et al.*, *op. cit.*, pp.16-21.

sencillo ó complejo, de acuerdo a los requerimientos específicos de cada programa.

En cuanto al segundo punto, es aconsejable considerar lo siguiente:

- Las características del sistema operativo, en lo que a manejo de archivos y de memoria se refiere.
- El tipo de memoria donde se almacenan los datos o la información, como sería la velocidad de acceso (mediana o alta) o el tipo de acceso.
- Los tiempos de acceso a los dispositivos.
- El tiempo de ejecución de un programa.

Medición del tiempo de ejecución de un programa. Tal vez este sea uno de los parámetros, más significativos parta medir la eficiencia del un algoritmo y depende de entre otros factores de los siguientes:

De la calidad y tipo del código que genere el compilador utilizado, en la construcción del código objeto.
De la cantidad, el tipo (tipo numérica o carácter) y la distribución inicial de los datos o de la información.
De la naturaleza y velocidad de las instrucciones de máquina utilizadas en la ejecución del programa,
De la complejidad de tiempo del algoritmo del programa.
De la velocidad de la computadora que se esté utilizando.

En este último apartado, uno de los factores que más se utiliza para evaluar el tiempo de ejecución de un programa, es el “tamaño” de la entrada, la cual es la longitud de elementos de que consta y generalmente se denota por la letra **n** y se acostumbra utilizar la notación **T(n)**, para el **tiempo de ejecución de un programa** con una entrada de tamaño **n.**

Se puede considerar a **T(n)** como el número de instrucciones ejecutadas en una computadora idealizada. Este tiempo de ejecución del programa, no depende sólo del tamaño de la entrada, si no que también es una función de ella. Se define también a **T(n)** como el tiempo de ejecución, considerándolo como el peor de los casos, esto quiere decir que es el máximo valor del tiempo de ejecución para entradas de tamaño n.

También existe otro tiempo como el $T_{promedio}(n)$, el valor medio del tiempo de ejecución para entradas de tamaño n y supone que todas las entradas son igualmente probables. Es difícil de evaluar este tiempo promedio, por lo que se utiliza el T(n) que es el tiempo de ejecución del peor caso, como medida principal de la complejidad del tiempo.

Existen algunas herramientas, que proporcionan una referencia en la velocidad de crecimiento, tomando en cuenta la cantidad de datos que se van a utilizar y son:

Notación asintótica **O (Mayúscula)**

Ω (Omega Mayúscula)

Estas notaciones se utilizan para hacer contemplar la velocidad de crecimiento de los valores de una función. Cuando se dice que **T(n) es O(*f*(n)),** se tiene que $f(n)$ es una cota superior para la velocidad de crecimiento de T(n), en tanto que para la cota inferior para la velocidad de crecimiento de **T(n) es Ω(g(n)).**

En general, se considera que es posible evaluar programas comparando sus funciones de tiempo de ejecución. Un programa con tiempo de ejecución es $O(n^2)$, es mejor que uno que tenga tiempo de ejecución $O(n^3)$, pero que para algunos casos, este tiempo de ejecución pueda depender de un factor constante debido a la naturaleza del programa mismo y con este factor no siempre se cumpla que $O(n^2)$ sea mejor que $O(n^3)$, por ejemplo, si el tiempo

de ejecución de un programa tarda 100 n^2 y otro programa que realiza la misma función con otro algoritmo tarda 5 n^3 (utilizando ambos las mismas unidades de tiempo como por ejemplo milisegundos).

En este ejemplo, aparentemente 100 n^2 es mejor que 5 n^3, paro si se evalúa, para tamaños de entrada *n < 20*, el segundo programa, a pesar de ser $O(n^3)$, es más rápido, que el primer programa que tiene $O(n^2)$, debido precisamente al factor por el cual se está multiplicando (100 o 5).

Pero para n = 20 los dos programas tienen el mismo valor de tiempo (40 000 unidades de tiempo) y a partir de n > 20, $O(n^2)$ es más rápido, que el programa de $O(n^3)$.

Si no tuviera esa factor siempre se cumpliría que $O(n^2)$ es más rápido, que el programa de $O(n^3)$, debido a que a medida que crece el tamaño de la entrada *n,* el programa de $O(n^3)$ requiere un tiempo mucho mayor que el programa que utiliza la función $O(n^2)$, debido, precisamente a la potencia a la cual esta elevado *n*.

En general, un programa se ejecutará más rápidamente si su función se apega más a una expresión lineal y que sean afectados por factores cercanos a la unidad.

También existe el criterio de que los mejores métodos cumplen con la función:

$n (\log_2 (n))$ (número de comparaciones).

Nótese que la velocidad de crecimiento es la que determina el tamaño del problema que se puede resolver en una computadora, por lo que se busca considerar programas cuyas velocidades de crecimiento sean las más óptimas o pequeñas posibles.

Calcular el tiempo de ejecución de un programa puede llegar a ser un proceso muy complejo, pero ya existe toda una teoría para aplicar algunos principios como es el de sumar y el de multiplicar en notación asintótica, como por ejemplo en la suma, se ocupa para la ejecución de una secuencia de pasos del programa y se toma el tiempo de la función ***O (Mayúscula)*** máxima de todas las que componen a esta secuencia, como el tiempo de ejecución de toda la secuencia.

Ya existen para algunos algoritmos muy comúnmente utilizados, como son los de clasificación (objeto de este tema) que en algunos de ellos, ya se ha realizado un estudio para obtener estas funciones de tiempo.

2.7. Análisis comparativo de las complejidades de los distintos métodos de ordenamiento

Los métodos que se trataron en este capítulo, ya son muy estudiados y por lo mismo sus funciones de comportamiento ya se tienen definidas, en seguida se mencionaran las más comunes de estos métodos.

Análisis de la Burbuja

Este método es muy sencillo de programar, puede llegar a ser muy rápido en cuanto a su ejecución, pero va a depender del número de datos denominado **n,** que va a clasificar, ya que a mayor cantidad de datos el método va siendo más lento.

El número de pasadas dependerá del número de datos, ya que como se menciono anteriormente en cada pasada se va colocando un elemento, a excepción de la última pasada, en donde se colocan dos elementos.

Esto es se va a tener **n-1** pasadas, generalmente el número de comparaciones es menor a **n-1,** una de las posibles ventajas de este método es que el número

de intercambios, se reduce en función de que los datos no vengan tan "desordenados", esto es que vengan muy cercanos a la posición que le correspondan, ya una vez clasificado, como se vio en el ejemplo, desde la tercera pasada, ya se encontraban clasificados, pero fue necesario realizar todas las pasada, sin embargo, ya no se hizo ningún intercambio.

Esto quiere decir que el número de intercambios va a depender de la distribución de los datos, no requiere prácticamente de espacio adicional de memoria para ejecutarse, ya que todo lo hace en el mismo arreglo o archivo y solo requerirá de algunas localidades de memoria adicionales, para guardar el elemento que se este intercambiando en esa pasada.

La función del comportamiento de este algoritmo, después de haber sido analizada por una gran cantidad de autores es:

$$n^2 / 4$$ [12]

Análisis de Selección Directa

Este método es muy sencillo en cuanto a la programación del algoritmo, de tamaño pequeño con respecto al código, rápido en ejecución siempre que se trate de "pocos" elementos a clasificar, ya que para encontrar al elemento más pequeño (o más grande) es muy rápido, cuando se trata de pocas llaves, en cuanto esta aumentan en número el método se tarda más en encontrar al elemento para seleccionarlo.

No es lo mismo encontrar un elemento (el más chico o el más grande) en un conjunto de diez elementos, que en uno de diez mil.

[12] Jorge Iván Euán Ávila y Luis Gonzaga Cordero Borboa , *op. cit.*, p. 152

Después de un análisis muy completo se ha encontrado que el total de comparaciones es de:

$$(n-1) + (n-2) + ... + 2 + 1$$

En la primera pasada se realizan n-1 comparaciones, para obtener el elemento deseado, el cual se elimina del conjunto; para la segunda se efectúan n-2 comparaciones y así sucesivamente.

Para que finalmente esta sucesión pueda ser representada como

$$\frac{(n^2 - n)}{2}$$ [13]

Siendo ***n*** el número de elementos de la entrada.

Una de las ventajas de este método es que casi no requiere de memoria adicional, ya que los elementos que se van seleccionando (el más grande o el más chico), van quedando en el mismo arreglo pero al final ya quedan clasificados, sólo necesitará de algunas variables adicionales para realizar los intercambios.

Análisis de Selección Repetitiva

Este método es uno de los más clásicos y estudiados por diferentes expertos en el tema, el número de comparaciones en cada uno de los grupos es la raíz cuadrada de **n-1**.

[13] Jorge Iván Euán Ávila y Luis Gonzaga Cordero Borboa , *op. cit.*, p. 143

Existen varias pasadas, siendo para la primera el número de comparaciones, que se obtiene de la multiplicación del número de grupos por el número de comparaciones con lo que se obtiene **n-1**.

En tanto que en las siguientes pasadas, el número de comparaciones es dos veces la raíz de **n - 1**, queda representado el número total de comparaciones como:

$$(n-1) + (n-1) * [\ 2\ \sqrt{n}\ - 1)\ ]$$ [14]

Al diferencia del método de Selección Directa, requiere de más memoria, ya que por cada nivel que tenga el árbol (dependiendo del número de raíz que se tome $\sqrt[2]{n}$, $\sqrt[3]{n}$, $\sqrt[4]{n}$, etc.), se requerirá de más nodos, aparte de un espacio igual al de los datos originales, para ir almacenando, los elementos "ganadores" ya clasificados, pero con la ventaja de ser mucho más rápido en su ejecución, sobre todo cuando el número de elementos aumenta.

Análisis de Torneo

En el nivel más bajo del árbol, también conocido como último nivel **u**, el número de comparaciones es **n/2**, en el nivel **u -1** es **n/4** y así sucesivamente hasta que se llega al nivel 2, donde sólo se requerirá de una comparación, por lo que el número de comparaciones en total en la primera pasada es:

n/2 + n/4 + n/8 + n/16 + …….+ 8 + 4 + 2 + 1, esto se comporta y se puede reducir como:

n -1

En las siguientes pasadas, el número de comparaciones, son tantas, como niveles tenga el árbol menos una, esto es porque en ya en la raíz no hay comparación.

---

[14] Jorge Iván Euán Ávila y Luis Gonzaga Cordero Borboa , *op. cit.*, p. 145

Y el número de comparaciones finalmente resulta:

$\log_2 (n)$ [15]

Las ventajas y desventajas de este método prácticamente son muy similares a las de Selección Repetitiva, de hecho es un caso particular de ella, donde sólo se consideran grupos de dos elementos.

Análisis del H*eap*

En el caso más critico, el borrado del H***eap***, se realiza en 2(n – 1) * d comparaciones, siendo n el número de elementos y d

$$d = |\log n| - 1$$

Para la construcción del H***eap*** el número de comparaciones en el peor de los casos es:

(n – 1) + 2 * d

Y el promedio de comparaciones que se realiza es de:

$n * \log_2 (n)$ [16]

Este método es uno de los más eficientes, ya que no ocupa prácticamente memoria adicional, ya que todo lo está haciendo en el arreglo o archivo original y ahí se realizan los intercambios, además como se trata de un árbol binario se puede almacenar en un arreglo de una dimensión y por sus características de ser binario, es muy sencillo el algoritmo de ir construyendo las estructuras del **H*eap*** y realizar los intercambios.

Análisis del SHELL

El número de comparaciones está en función del tamaño de los incrementos usados y de su secuencia.

---

[15] Jorge Iván Euán Ávila y Luis Gonzaga Cordero Borboa , *op. cit.*, p. 148

[16] **idem**, *op. cit.*, p. 150

Unas secuencias recomendables para los valores h son:

ele =2 entonces h_1 = 1.72 $\sqrt[3]{n}$ y h_2 = 1, utilizando estos valores, el número de comparaciones es proporcional a $n^{5/3}$. Recordando que ele, es la variable que indica cuantos elementos tiene el arreglo donde se guardan los incrementos h. En caso de ocupar más de dos incrementos la secuencia de valores es h(k) = 2^k - 1 para 1 <= k <= log (n), el número de comparaciones es:

$n^{3/2}$ [17]

Este método no requiere prácticamente de memoria adicional, todo lo realiza sobre el mismo arreglo, y algún espacio de memoria, para las variables que se utilizan para los intercambios.

TABLA COMPARATIVA DE ALGUNOS METODOS DE CLASIFICACIÓN

Nombre del método	Uso de memoria adicional, salvo para variables auxiliares, para realizar los intercambios.	Función de comportamiento, considerando el número de comparaciones	Sencillez en la programación del algoritmo	Velocidad de ejecución	Número aproximado de elementos con los que se mantiene su rápida ejecución.*
Burbuja	Baja	(n*n)/4	Baja	Baja	Bajo
Inserción Directa	Baja	(n*n-n)/2	Mediana	Mediana	Bajo
Selección Directa	Baja	≈ n* n ó	Mediana	Mediana	Bajo

[17] Jorge Iván Euán Ávila y Luis Gonzaga Cordero Borboa , *op. cit.*, p. 159

		$\frac{(n^2-n)}{2}$			
Selección Repetitiva	Mediana	$n\sqrt{n}$ $n\sqrt[3]{n}$	Mediana	Alta	Alto
Torneo	Alta	$n \log_2 n$	Alta	Alta	Alto
Heap	Alta	$n \log_2 n$	Alta	Alta	Alto
Shell	Baja	$n^{3/2}$	Alta	Alta	Alto

* Son muy relativos estos parámetros de alto, mediano o alto, pero va a depender en buena medida de la función de comportamiento

Es posible graficar por dos caminos los métodos de clasificación:

A través de las funciones que definen el comportamiento de los métodos.
Por pares ordenados (**x**, **y**), donde se tendrán los ejes coordenados siendo las abscisas (**X**) el número de elementos y las ordenadas (**Y**) el tiempo. Para obtener estos puntos se ejecutan varias veces cada método, por ejemplo, realizar 10 ejecuciones de cada método con 5, 10, 15, 20, 25, 30, 35, 40, 45 y 50 llaves (datos), que pueden ser generadas aleatoriamente. Los cuales deberán utilizar las mismas llaves y en el mismo orden en que fueron generadas, para ser más homogéneas las ejecuciones y poder realizar el análisis del tiempo de ejecución de los algoritmos.
El tiempo se puede tomar a través del reloj del equipo de cómputo, aunque este es sólo aproximado, ya que lo da en segundos y para esta cantidad pequeña de llaves, generalmente da cero, por lo que se recomienda utilizar una función de retraso.

Cuando el número de llaves es más grande se puede tomar el tiempo directamente de lo que tarda en ejecutarse el método, un programa en *C*, puede comparar los datos tomados de *n* (el tamaño de la entrada) con su *t*, que es el tiempo que tardo en ejecutarse con esa entrada.

A continuación, se presentan algunas salidas de un programa, que fue desarrollado en C en diferentes escalas. Este tipo de gráficas, permiten tener más elementos de decisión, para ver hasta que número de elementos conviene utilizar uno u otro de los métodos vistos con anterioridad.

```
GRÁFICA DEL COMPORTAMIENTO DE METODOS DE CLASIFICACIÓN.
Proporcione la escala en x 10.000000
Proporcione la escala en y 5.000000

       GRAFICA DEL COMPORTAMIENTO DE LOS SIGUIENTES METODOS:
    Nombre del metodo  Caracter con que se representa  funcion.
       Burbuja                %%%%%%%%%                (n*n)/4
       Seleccion Directa       *********                  (n*n-n)/2
       Shell                  &&&&&&&&&           n elevado a la 3/2
       Respuesta ideal        #########                n*ln(n)
                     t es tiempo y n es el numero de elementos
 t
  |                                                 #
7=                                          &
  |                                                 *          %
  |                                                    #
  |                                        &                   %
  |                                                * #
6=
  |                                        &                   %
  |                                                   #
  |                                                 *          %
  |                                        &        #
5=                                                             %
  |                                        &        #
  |                                                          %
  |
  |                                     &        #      %
4=
  |                                  &        #      %
  |                                                 *
  |                               &         #      %
  |                                                 *
3=                             &         #      %
  |
  |                           &         # *    %
  |                                       %
  |                        &         # *
2=                                      %
  |                     &        #   * %
  |                                    %
  |                  &         # %
  |                            #
1=               &          % %
  |           &          # *
  |       &          # *
  |     &      % # *
  |    &% % %    *
-#-#-------|---------|---------|---------|---------|---------|--n
          1         2         3         4         5         6
```

Figura 2.46. Gráfica del comportamiento de métodos de clasificación X=10, Y=5

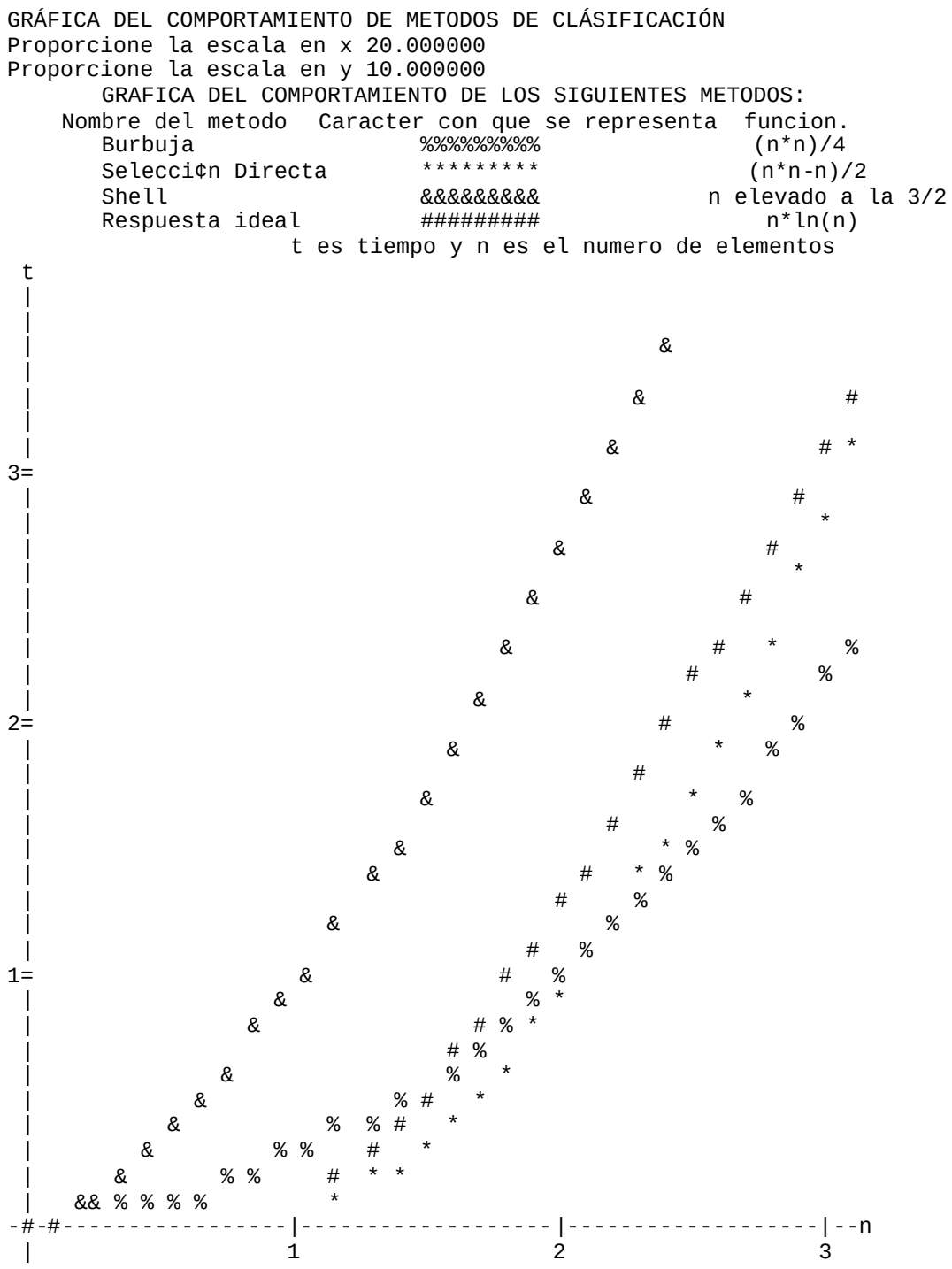


Figura 2.47. Gráfica del comportamiento de métodos de clasificación X=20, Y=10

```
GRÁFICA DEL COMPORTAMIENTO DE METODOS DE CLÁSIFICACIÓN.
Proporcione la escala en x 20.000000
Proporcione la escala en y 15.000000

       GRAFICA DEL COMPORTAMIENTO DE LOS SIGUIENTES METODOS:
    Nombre del metodo  Caracter con que se representa  funcion.
       Burbuja                 %%%%%%%%%                (n*n)/4
       Selecci¢n Directa       *********                  (n*n-n)/2
       Shell                   &&&&&&&&&           n elevado a la 3/2
       Respuesta ideal         #########                n*ln(n)
                     t es tiempo y n es el numero de elementos
```

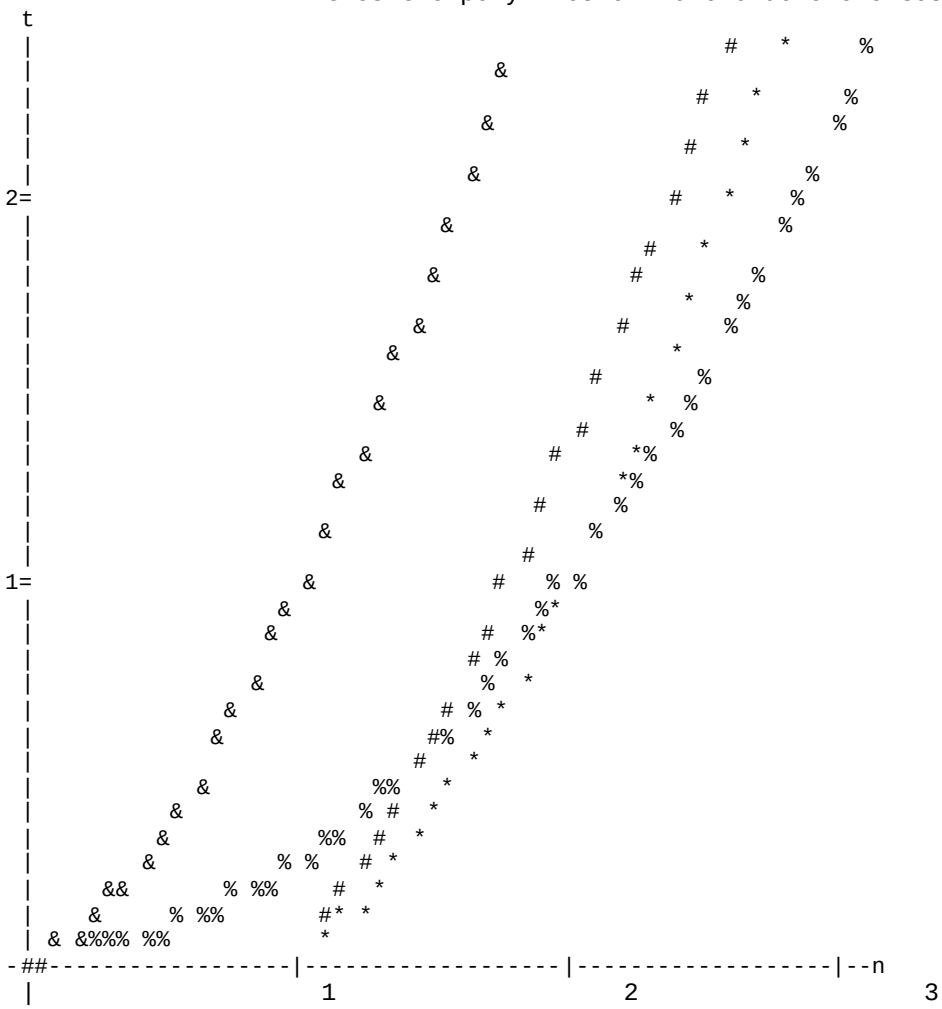


Figura 2.48. Gráfica del comportamiento de métodos de clasificación X=20, Y=15

Las graficas que se presentan en las figuras anteriores indican con diferentes símbolos la respuesta de cada uno de los métodos y al principio cuando se tienen pocos elementos van muy parejos en su tiempo de respuesta, pero conforme aumentan las llaves o datos a clasificar, el método de la burbuja va tardando más en ejecutarse, en tanto que el Shell, es el que más rápido se ejecuta, sobre todo cuando el número de elementos aumenta.

Se utilizaron estos tres métodos, por ser muy similares en lo que se refiere a la ocupación de memoria adicional, los tres lo hacen sobre el mismo arreglo.

A través de este tipo de comparaciones se puede tener una idea de qué algoritmo conviene utilizar, dependiendo del número de entradas, que aunque es uno de los factores principales, también lo es la distribución con que vienen los datos.

Los métodos de clasificación, no tan sólo siguen estando vigentes si no que cada vez van tomando mayor importancia, a pesar de que ya han pasado varios años, desde que se formalizó su creación.

Esto es debido a que cada vez se va haciendo más necesaria la información y los datos, así como su manejo eficiente ya que en esta época, la materia prima, ya no tan solo lo son los recursos naturales sino también el conocimiento, con lo cual conlleva a su manejo eficiente.

Este manejo implica su actualización y consulta para la toma de decisiones y para poderlo hacer, es necesario, realizar en la mayoría de los casos una clasificación.

En este tema se trataron los métodos más conocidos, pero cabe mencionar que sólo es una muestra algunos de ellos, ya que existen muchos más, tanto internos como externos.

Bibliografía del tema 2

AHO, Alfred V., *et al.,Estructuras de Datos y Algoritmos*, Estados Unidos, Addison-Wesley Iberoamericana, 1988, pp. 438.

EUÁN ÁVILA, Jorge Iván y CORDERO BORBOA, Luis Gonzaga, *Estructuras de datos*, México, Limusa, 1989, pp. 219.

KNUTH, Donald E., El arte de programar ordenadores, Volumen III Clasificación y Búsqueda, (impresión.), España, Reverté, 1987, 776 pp.

Actividades de aprendizaje

A.2.1. Elabora un mapa conceptual del tema.

A.2.2. A partir del código del programa, donde se implanto el algoritmo de burbuja, realiza el pseudocódigo correspondiente.

A.2.3. A partir del pseudocódigo del algoritmo del método de Inserción Directa, codifícalo en un programa en **C.**

A.2.4. A partir del código del programa donde se implanto el algoritmo de Selección Directa, realiza el pseudocódigo correspondiente.

A.2.5. Modifica el pseudocódigo del algoritmo de Selección Repetitiva, donde se empiece con índices de los arreglos en **uno** y no en **cero,** como está hecho el que se presentó.

A.2.6. A partir del método de Selección Repetitiva modifícalo para que ahora la clasificación sea en forma descendente, solo en pseudocódigo. Empezando con los índices en **ceros.**

A.2.7. Codifica en **C**, el algoritmo de Selección Repetitiva para que los datos queden en forma descendente y empezando en índices en **ceros.**

A.2.8. Modifica el pseudocódigo, del algoritmo de Selección Repetitiva, de tal forma, que se contemple el caso de cuando la raíz cuadrada del número de elementos **n**, no sea un número entero exacto (esto es que tenga fracciones), por ejemplo **n** = 10, su raíz cuadrada sería 3.1622.

A.2.9. Realiza el pseudocódigo del algoritmo de Selección Repetitiva, cuando se tenga una raíz cúbica de **n** **(** $\sqrt[3]{n}$).

A.2.10. Retoma el pseudocódigo del método de Selección Repetitiva, modifícalo, para obtener el del método de **Torneo**.

A.2.11. Encuentra el pseudocódigo del método de **Torneo**, para contemplar el caso de que no se trate de un árbol binario completo, esto es que el número de elementos **n**, que se van a clasificar no sean (2^n -1), con lo cual no se tendrán grupos completos de dos elementos y va a ser necesario, marcarlos con infinito (∞).

A.2.12. Realiza el pseudocódigo, del método del **H*eap***.

A.2.13. Utiliza el método del Shell para clasificar los siguientes elementos en forma ascendente:

4, 6, 3, 2, -9, 1, 9, 7, 8, 5, 0

los incrementos **h** que se van a utilizar son:

6, 3, 2, 1 o 5, 3, 2, 1

Observe los resultados y vea que tomando diferentes incrementos, puede o no clasificarse correctamente los datos.

A.2.14. Utiliza el método del Shell, con los mismos datos, pero con la siguiente distribución. Clasifícalos también en forma ascendente:

0, 4, 6, 3, 2, -9, 1, 9, 7, 8, 5

los incrementos **h** que se van a utilizar son:

6, 3, 2, 1

A.2.15. Modifica el código, que se presenta a continuación del método del Shell, de tal forma que se utilice el arreglo **incr** con los incrementos de h, como se vio en el pseudocódigo visto en este tema y no que los vaya dividiendo como viene en el siguiente código. Nótese que se tomaron los tiempos con un retraso de 10 unidades para poder conocer el tiempo que tarda en ejecutarse el método con algún número de datos; estos parámetros van a ser utilizados en un ejercicio posterior.

```
void shell(int a[], int num, int z)
{
   int c=0,i=0,k,j=0,h,n,m;;
   printf("\n\t**** METODO DE SHELL *****\n\n");
```

```
printf("\n\t*** los elementos desordenados son ***\n");
for (k=0;k < num; k++)
       printf (" %d ",a[k]);
printf("\n\n");
h=num;
principio = clock();
while (h > 0)
{
    h=h/2; //Se va dividiendo entre dos los incrementos y en
                //el pseudocodigo estos valores estan en un arreglo
    for (j=h; j <= num; j++)
    {
     delay(10); // Aqui se retrasa 10 unidades
      i=j-h;
      k=a[j];
      do
      {
         if (k < a[i])
         {
           a[i+h]=a[i];
           i=i-h;
         }
         else
           break;
      } while (!(i<=0));
      a[i+h]=k;
    }
 }

fin = clock();
num=num-1;
printf("\n\t*** los elementos ordenados en forma");
printf(" asecendente son ***\n");
for (k=0;k <= num; k++)
       printf (" %d ",a[k]);
nele[2][z]=num+1;
ntiempo[2][z]=((fin - principio)) / CLK_TCK ;
printf("\n\n");
printf("\n\t*** El tiempo de ejecucion en decimas de segundos\n");
```

```
   printf("\t\ten forma ascendente fue de: %7.5f ",((fin - principio))
/ CLK_TCK);
   printf("****");
}// fin de shell
```

El resultado de la corrida del programa es:

```
       **** METODO DE SHELL *****
       *** los elementos desordenados son ***
 86  39  62  77  77  86  74  55  82  86  3  88  44  40  9

       *** los elementos ordenados en forma ascendente son ***
 3  9  39  40  44  55  62  62  74  77  77  82  86  86  86

       *** El tiempo de ejecucion en decimas de segundos
              en forma ascendente fue de: 0.54945 ****
```

A. 2.16. Para el método del *Quick*, encuentra su pseudocódigo, además realiza el programa en C.

A.2.17. Realiza un programa en C, similar al que muestra las gráficas de las figuras 2.38 a la 2.40, en donde se den las funciones del comportamiento de los métodos y las grafique.

A.2.18. Realiza otro programa en C, para que se ejecute con varios valores de elementos y tomen sus valores en cuanto al tiempo de ejecución, para por lo menos 3 métodos y que sean los mismos que se escogieron en el punto anterior. Almacena estos valores (tiempo y número de elementos), ya sea en un archivo o en un arreglo de una dimensión, para que con otro programa, los grafiques a partir de estos puntos. Después compara ambas graficas.

A.2.19. Realizar un programa en C, que tenga como entrada un archivo o arreglo, donde cada registro o elemento del arreglo (puede ser arreglo de estructuras), contendrá los datos de cada alumno que se va a inscribir y tiene los siguientes campos en su formato:

1. Número de cuenta de ocho dígitos numéricos.

2. Nombre compuesto de apellido paterno, materno y nombres de treinta caracteres alfabéticos.
3. Clave de la carrera de tres posiciones numéricas.
4. Clave de la materia también de tres posiciones numéricas.
5. Número de grupo de dos posiciones numéricas.

Como salidas serán las listas que se les proporciona a los profesores para su control escolar. Estas listas, deberán de estar en orden alfabético los nombres de los alumnos y se desplegarán los nombres de los alumnos, la clave de la materia, la clave de la carrera y el número de grupo. .

Cuestionario de autoevaluación

1. ¿Cuál es la diferencia entre métodos de clasificación internos y de los externos?
2. ¿Cuáles son los grupos en qué se pueden dividir los métodos de clasificación internos?
3. ¿Qué es clasificar (ordenar)?
4. ¿Cómo se puede definir la clasificación en forma ascendente?
5. ¿Qué es la forma descendente de clasificación?
6. ¿Cuál es el número máximo de elementos que pueden tener los grupos de elementos del método de Torneo?
7. ¿De qué tipo de árboles requiere el *heap*?
8. El Shell, ¿de que tipo de árbol necesita?
9. ¿Qué valor se recomienda para iniciar los incrementos h, en el método del Shell?
10. ¿Cómo se van disminuyendo los valores de h?

Examen de autoevaluación

Relaciona la columna de la izquierda con la de la derecha y coloca la respuesta que consideres correcta dentro del paréntesis.

( ) 1. Es uno de los métodos de intercambio.	a) Intercambio.
( ) 2. Son métodos que utilizan árboles binarios	b) Métodos internos y también externos
( ) 3. Estos métodos, ocupan principalmente el mismo espacio de memoria original	c) Burbuja
( ) 4. El *quick* pertenece al grupo de:	d) n * ln(n)
( ) 5. El *heap* es del grupo de	e) Shell y *heap*
( ) 6. El método de distribución pertenece a	f) (n * n)/4
( ) 7. La función de comportamiento ideal de un método de clasificación es:	g) (n * n - n) / 2
( ) 8. La burbuja utiliza la función	h) Sistema operativo
( ) 9. La función de selección directa es:	i) Torneo y *heap*
( ) 10. Uno de los elementos que interviene en la velocidad de ejecución de un programa	j) Selección

TEMA 3. MÉTODOS DE BÚSQUEDA

Objetivo particular.

Al concluir este tema, el alumno será capaz de:

Conocer los métodos más importantes de búsqueda y aplicar el más conveniente al conjunto de datos que se encuentran, ya sea en memoria principal o en la memoria secundaria, así mismo manejará las funciones de dispersión.

Temario detallado

3.1. Búsqueda secuencial

3.2. Búsqueda binaria

3.3. Búsqueda mediante transformación de llaves (Hashing)

- 3.3.1. Función *hash*
- 3.3.2. Resolución de colisiones

3.4. Árboles binarios de búsqueda

Introducción

Una de las funciones que con mayor frecuencia se utiliza en los sistemas de información, es el de las consultas a los datos, se hace necesario utilizar algoritmos, que permitan realizar búsquedas de forma rápida y eficiente.

La **búsqueda,** se puede decir que es la acción de recuperar datos o información, siendo una de las actividades que más aplicaciones tiene en los sistemas de información, más formalmente se puede definir como “La operación de búsqueda sobre una estructura de datos es aquella que permite localizar un nodo en particular si es que éste existe”[18]

Una posible clasificación de las búsquedas puede ser:

[18] Jorge Iván Euán Ávila y Luis Gonzaga Cordero Borboa , *op. cit.*, p. 181

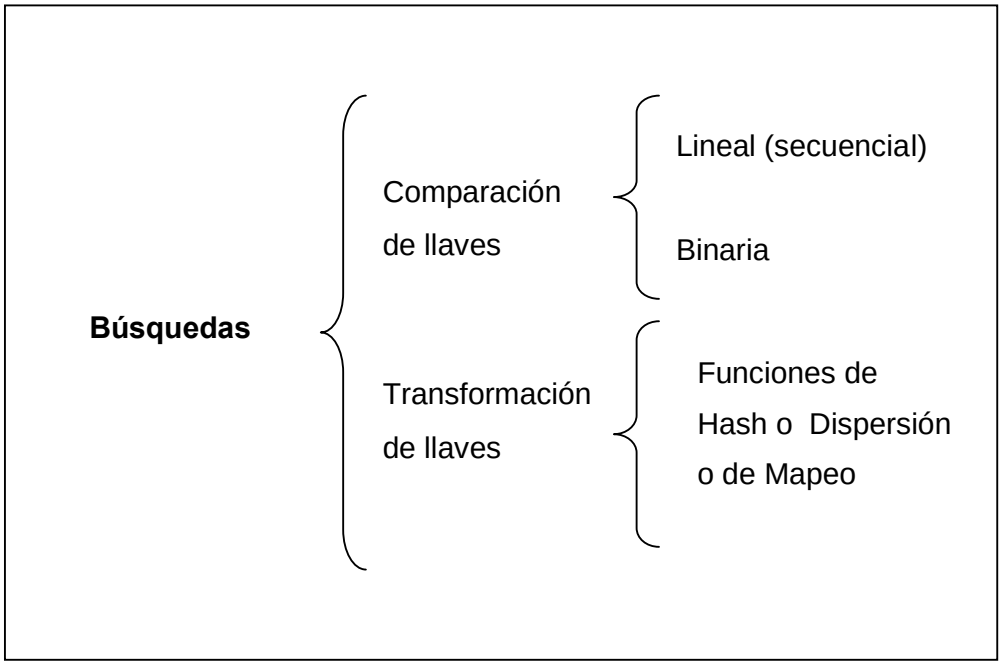


Figura 3.1. Clasificación de las búsquedas

Siendo la llave el campo por el cual se va a realizar la búsqueda. La búsqueda por comparación de llaves son algoritmos bastante sencillos, en donde se van comparando precisamente las llaves, pero pueden a llegar a consumir mucho tiempo cuando se tiene un gran número de ellas, en tanto que el de transformación de llaves los algoritmos deben de ser mucho más analizados, ya que transforma las llaves por varios métodos, indicando en que posición del arreglo o del archivo (acceso directo) lo va a almacenar o a recuperar.

3.1. Búsqueda secuencial

Este tipo de búsqueda consiste en examinar, a partir del primer elemento y de uno en uno, hasta encontrar el dato buscado o bien llegar al final de la lista que puede estar almacenada en archivo o arreglo.

En este tipo de listas los elementos pueden o no estar clasificados, ya que se empieza a comparar de uno en uno los elementos de la lista y no importa su orden para realizar la búsqueda, salvo para el tiempo de ejecución.

Si el elemento que se está buscando, se encuentra al inicio de la lista, este tiempo, sería muy corto, pero si se encuentra al final, va a tardar más y si el elemento que se desea buscar, no se encuentra en la lista, se hizo necesario, recorrer toda la lista, para darse cuenta que no está en ella.

Y si se le aumenta a esto, que el número de elementos en la lista puede ser del orden de cientos o miles, va a hacer mucho más tardado su ejecución.

Esta búsqueda tiene la ventaja de tener una fácil programación de su algoritmo.

El pseudocódigo del algoritmo es el siguiente:

```
inicio
        i = 0
        bandera = 0
        mientras i < n
                si k [ i ] = x
                inicio
                        desplegar “búsqueda exitosa y el dato es” x
                        desplegar  “y se encuentra en la posición”  i
                        i = n
                        bandera = 1
                else
                        i = i + 1
                fin
        fin
        si bandera = 0
                desplegar “no se encontró el dato en la lista”
fin
```

Para este pseudocódigo, se considera que se tienen n nodos (conjunto de uno o varios campos) en una lista como:

$\mathbf{n}_0$, $\mathbf{n}_1$, $\mathbf{n}_2$, ... $\mathbf{n}_{n-1}$

Cuyas llaves son: K_0, K_1, K_2, ... K_{n-1}, donde n es $\geq$ 0 y **x** es el valor del dato que se está buscando

Análisis de la búsqueda secuencial

El número de comparaciones tanto en el caso promedio como en el caso más crítico, es un parámetro importante para medir la eficiencia del método. El número promedio de comparaciones en la búsqueda de un elemento es:

n / 2

En el peor de los casos, cuando no se encuentra el elemento, es necesario recorrer toda la lista y entonces se realizan las siguientes comparaciones:

n

Ejemplos:

Supón que se tienen los siguientes datos (no forzosamente clasificados):

Posición dentro del arreglo y/o archivo	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
	1,	3,	9,	4,	2,	5,	7,	8,	0,	11,	15,	13,	12

1

Se desean encontrar los siguientes datos:

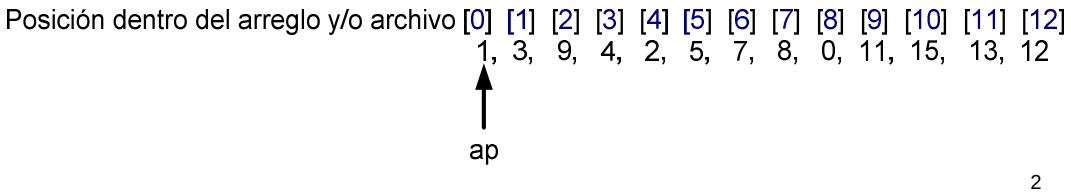
1

8

12

16

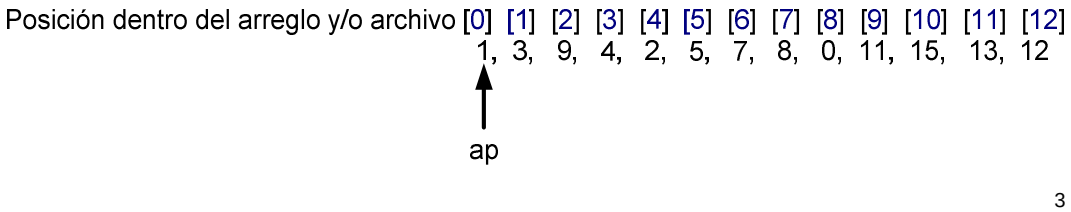
Solución A)



2

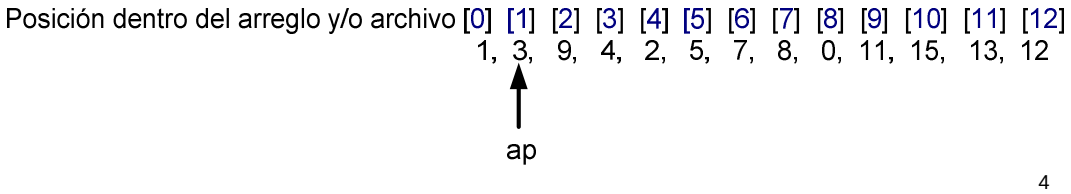
En la primera comparación se encuentra el elemento 1 (uno) que está en la posición 0 (cero).

Solución B)

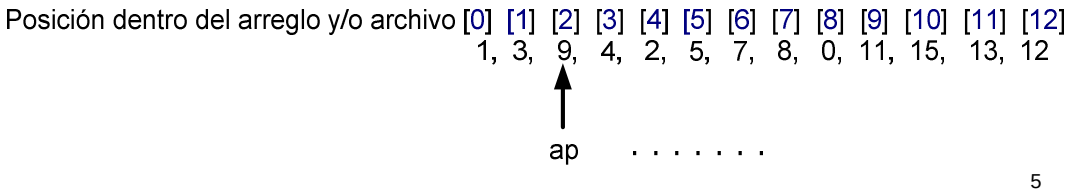


3

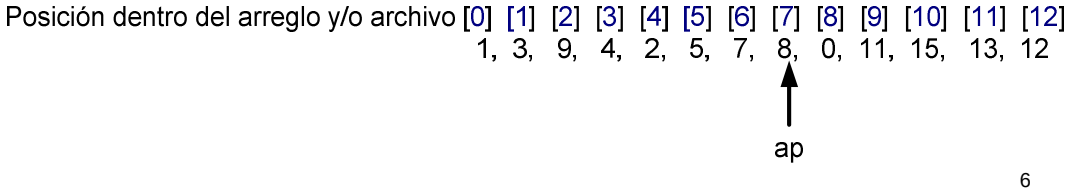
Se empieza a comparar desde la posición 0 (cero) y se va avanzando de uno en uno la posición, realizando la comparación, hasta que se encuentra el elemento buscado o termina la lista de datos.



4



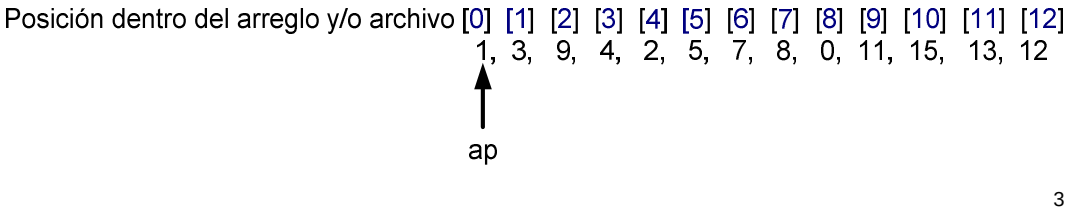
5



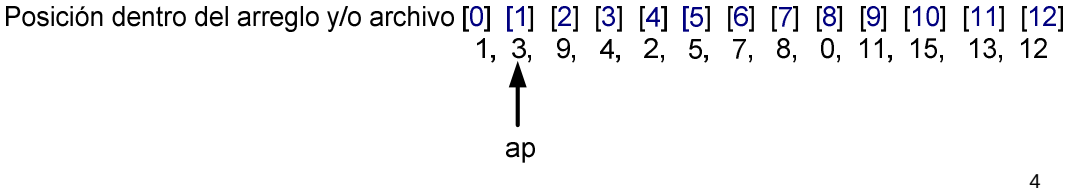
6

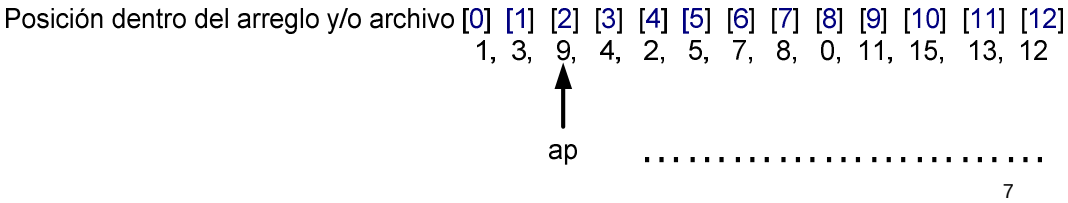
En la octava comparación se encuentra el elemento 8 (ocho) que está en la posición 7 (siete).

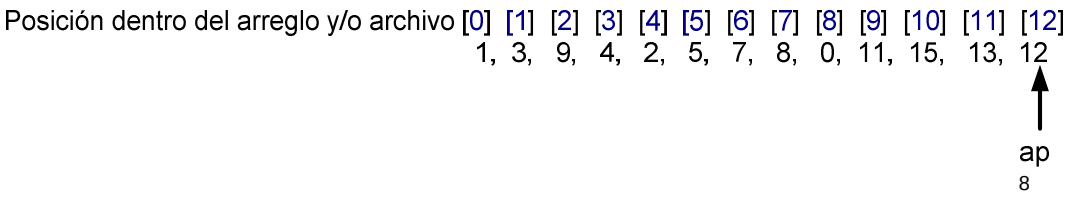
Solución C)



Se empieza nuevamente a comparar desde la posición 0 (cero) y se va avanzando de uno en uno la posición, realizando la comparación, hacia la derecha hasta que se encuentra el elemento buscado o termina la lista de datos.

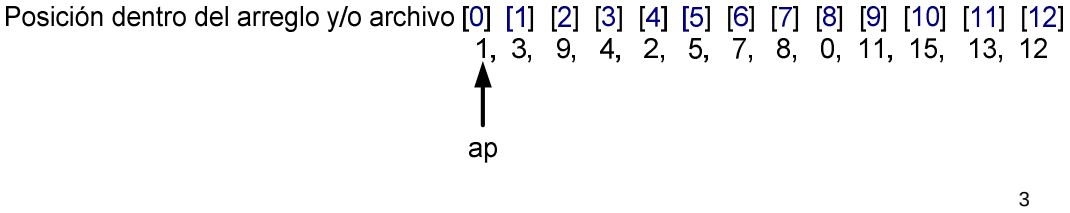




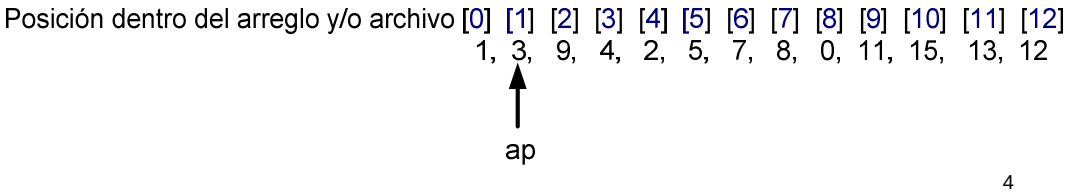


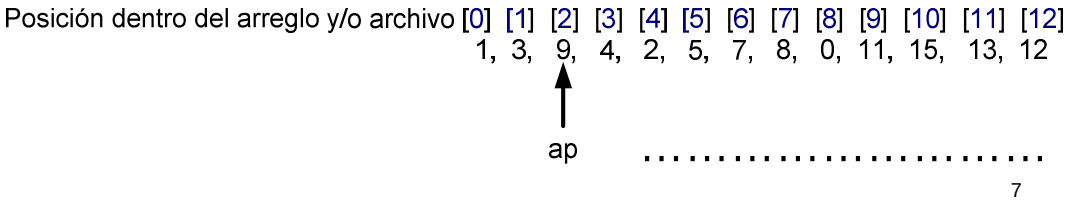
El elemento 12 (doce) está en la posición 12 (doce), después de trece comparaciones.

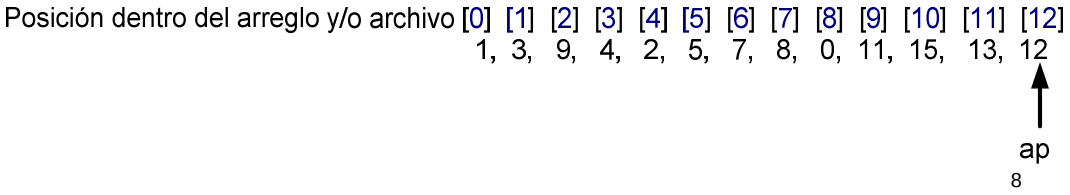
Solución D)



Se empieza nuevamente a comparar desde la posición 0 (cero) y se va avanzando de uno en uno la posición, realizando la comparación, hacia la derecha hasta que se encuentra el elemento buscado o termina la lista de datos.







El elemento 16 (dieciséis) **no se encuentra** en la lista, después de trece comparaciones y de haber recorrido toda la lista.

3.2. Búsqueda binaria

La búsqueda Binaria o por Bisección no representa mucha dificultad para la programación de su algoritmo y además, es muy rápida su ejecución.

Este algoritmo requiere que los elementos de la lista, sobre la que va a actuar, estén clasificados, ya sea en forma ascendente o descendente, cada elemento de la lista puede tener varios campos. La lista se considera que empieza a almacenar sus elementos en la posición **cero.**

Va a utilizarse tres apuntadores, uno en la primera posición de la lista que se le denominara **LI**, para efectos de la explicación, otro en la última conocido como **LS** y el que apunte en la parte central, el cual se obtiene de la suma de **LS mas LI** entre dos (LI + LS/ 2) y tomando la parte entera, el cual se le llamará **M.**

A diferencia de la **Búsqueda Secuencial**, aquí el número de comparaciones no se comporta en forma lineal, ya que procede a realizar los siguientes pasos:

- Dividir la lista en dos partes, al determinar el elemento central de dicha lista, con lo que se iniciará el apuntador **M.**
- Comparar el valor del elemento buscado con el central.
- Si resultan ser iguales, las búsquedas termina con éxito, indicando en qué posición se encontró y cuáles son los datos que están en esa posición.
- En el caso de no ser iguales, se redefinen la posición de alguno de los apuntadores de los extremos (**LI** o **LS**), dependiendo del valor del elemento central, sea mayor o menor que el buscado. Por ejemplo, suponiendo que la lista está clasificada en forma ascendente, se pueden presentar los siguientes casos:

    - Que el elemento buscado sea mayor que el elemento central (apuntado por **M**), entonces se desechará la primera mitad de la lista ya que por estar clasificada en forma ascendente se sabe que ahí no va a estar el elemento, con lo que se moverá el apuntador de inicio que es **LI** a la posición siguiente a donde está el elemento central que esta apuntado por **M.** Procediendo

a considerar esta nueva lista, que es la segunda mitad, como la nueva lista.

- Que el elemento buscado ahora sea menor que el elemento apuntador por **M**, entonces, se desechará la segunda mitad de la lista y el apuntador que se actualizará será entonces **LS**, que es el que apuntaba al último elemento y ahora se moverá a una posición inmediata anterior a donde está apuntando **M.**

En ambos casos se aplicaran nuevamente los puntos anteriores del método, desde el primer paso, hasta que el elemento sea encontrado o bien que la lista que vaya quedando como resultado, quede vacía.

El pseudocódigo del algoritmo es el siguiente:

```
inicio
        LI = 0
        LS = n – 1
        mientras   LI  ≤LS
                M = Parte Entera (LI + LS) / 2)
                si  dato <  lista [ M ]
                        LS = M -1
                en caso contrario si dato > lista [ M ]
                        LI = M +1
                en caso contrario
                        desplegar “el elemento buscado esta en la posición”, M
                        break
                fin
        desplegar “el elemento” dato “no se encontró en la lista”
fin
```

Dato, es el valor que se va a buscar en la lista; **lista**, es donde están los elementos clasificados, **LI** es el límite inferior, **LS** es el límite superior, **M** es el apuntador de en medio y **N** el número de elementos de la lista.

Análisis de la Búsqueda Binaria

Este método en cada iteración el intervalo de búsqueda se divide por mitad, por lo que el número de comparaciones tiende a:

$\log_2 n$

Ejemplos:

Se tienen los siguientes elementos de una lista, clasificada en forma ascendente:

Posición dentro del arreglo y/o archivo	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
	1,	2,	3,	4,	6,	7,	9,	12,	14,	16,	17,	19,	21

1

Se desean buscar los elementos (dato):

9

3

21

18

Solución A)

Posición dentro del arreglo y/o archivo	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
	1,	2,	3,	4,	6,	7,	9,	12,	14,	16,	17,	19,	21
	↑						↑						↑
	LI						**M**						**LS**

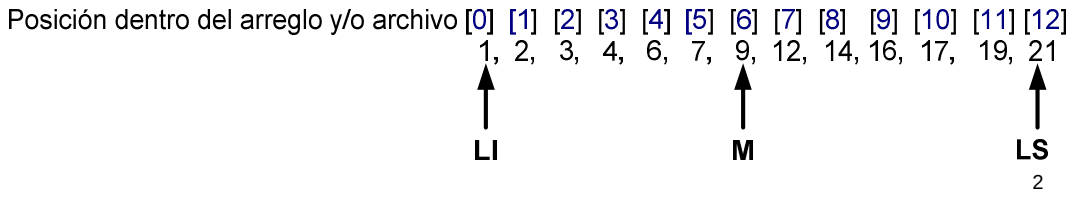
2

M= Parte Entera ((LI + LS) / 2)

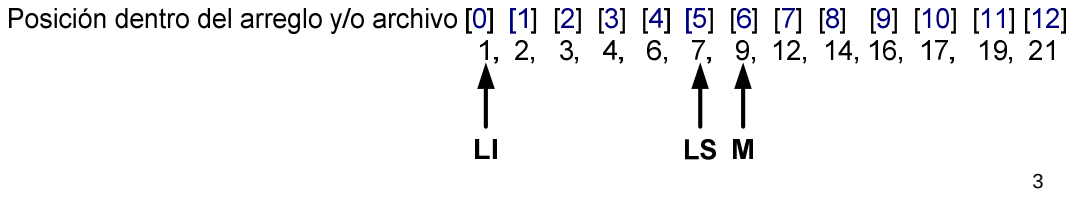
M= Parte Entera ((0 + 12) / 2) = 6

El elemento 9 (nueve) se encuentra en la posición 6 (seis) y se encontró en la primera comparación, ya que el valor del elemento dato es igual al valor apuntado por M.

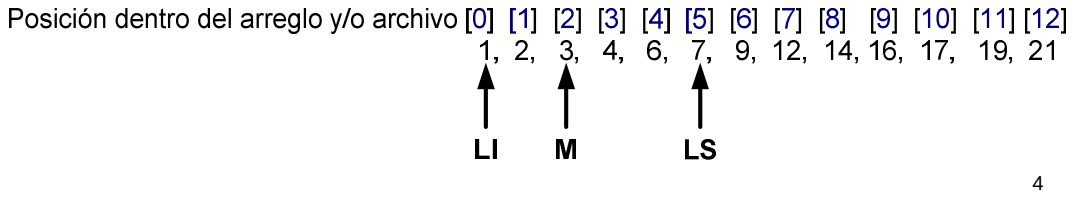
Solución B)



El elemento apuntado por M (9) es mayor al valor del elemento dato (3) y como la lista se encuentra clasificada en forma ascendente, entonces se va a desechar la segunda mitad de ésta.



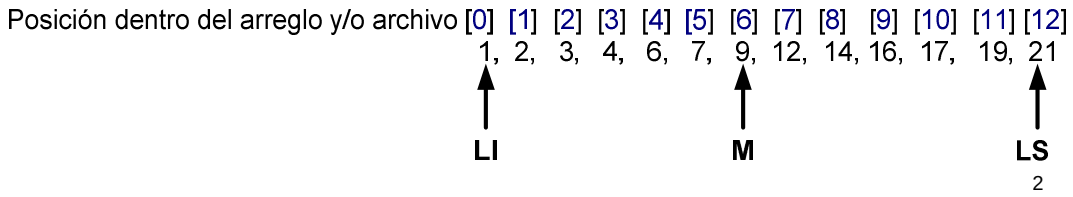
Se realiza la actualización del apuntador LS, que se va a mover una posición antes de M.



M= Parte Entera ((0 + 5) / 2) = 2

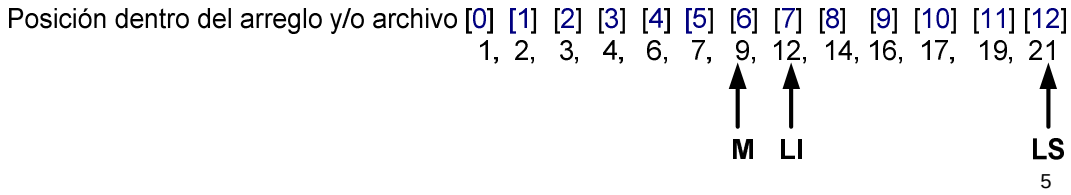
Se vuelve a calcular el valor de M y se realiza la comparación y se ve que el valor del dato que es un 3 (tres), es igual al valor donde apunta M, por lo que el elemento 3 (tres), está en la posición 2 (dos), realizándose dos comparaciones.

Solución C)

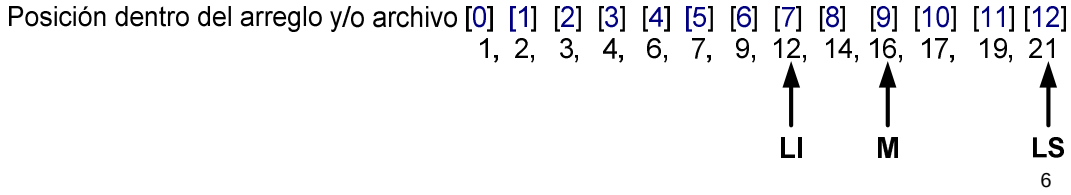


M= Parte Entera ((0 + 12) / 2) = 6

El elemento apuntado por M (9) es menor al valor del elemento dato (21) y como la lista se encuentra clasificada en forma ascendente, entonces se va a desechar la primera mitad.



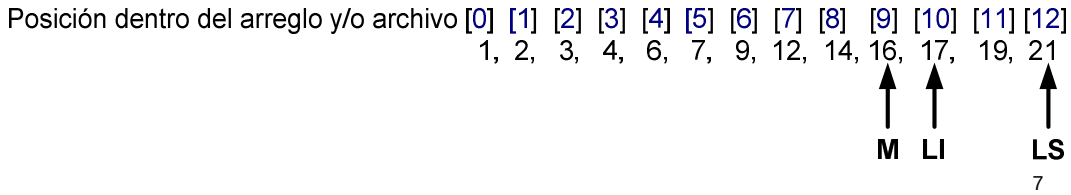
LI, se mueve una posición después de M



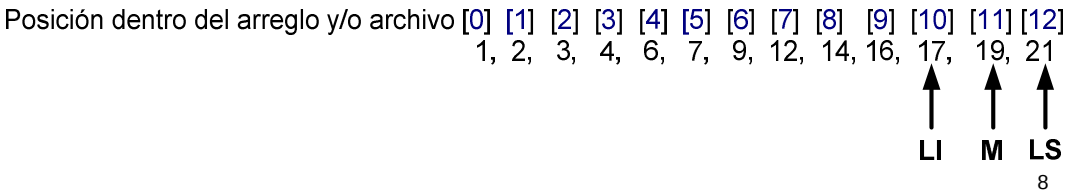
Se calcula nuevamente M como:

M= Parte Entera ((7 + 12) / 2) = 9

y como el valor que esta apuntado por M (16) es menor que el valor del dato (21), se descarta la primera mitad de esta nueva lista.



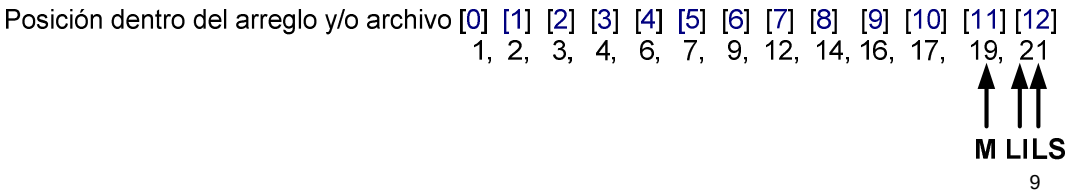
Por lo que se mueve el apuntador de LI, una posición después de M.



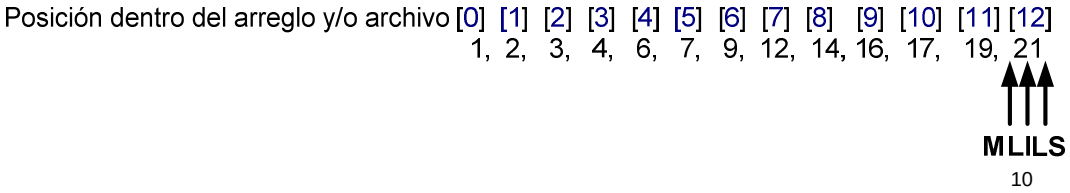
Se calcula el valor de M, con los nuevos valores de LI y LS como:

M= Parte Entera ((10 + 12) / 2) = 11

Como el dato (21) es mayor que el valor que apunta M (19) se desecha la primera mitad.



Se actualiza la posición de LI un lugar después de M.

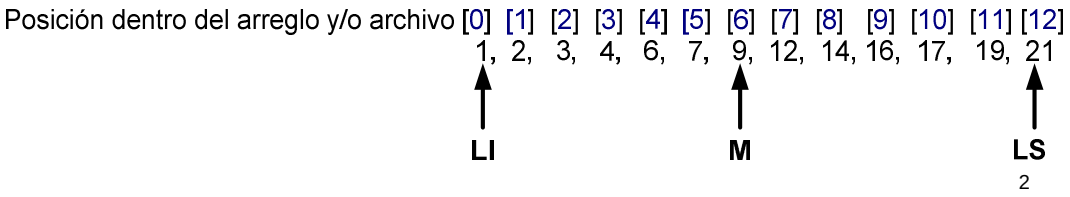


Se calcula nuevamente M:

M= Parte Entera ((12 + 12) / 2) = 12

Los tres apuntadores LI, LS y M, apuntan en esta ocasión al mismo elemento que se encuentra en la posición 12 (doce) y el elemento 21 (veintiuno) se encuentra después de la cuarta comparación en la posición 12 (doce)

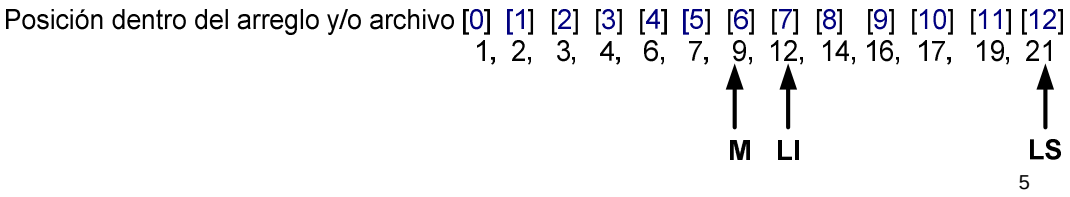
Solución D)



2

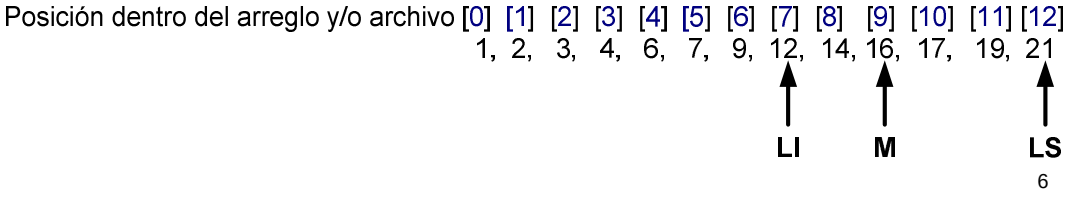
M= Parte Entera ((0 + 12) / 2) = 6

El elemento apuntado por M (9) es menor al valor del elemento dato (18), se va a desechar la primera mitad.



5

LI, se mueve una posición después de M

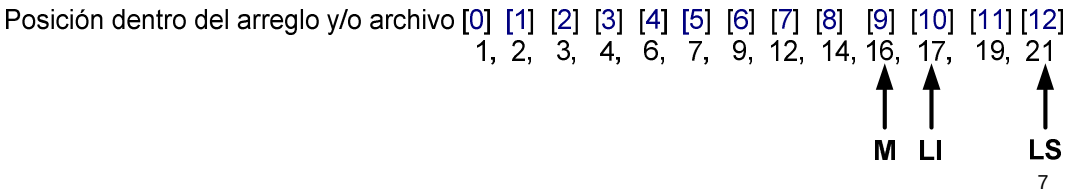


6

Se calcula M como:

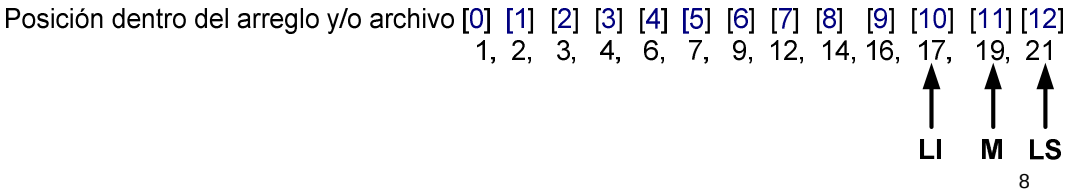
M= Parte Entera ((7 + 12) / 2) = 9

Como el valor que está apuntado por M (16) es menor que el valor del dato (18), se descarta la primera mitad de esta nueva lista.



7

Por lo que se mueve el apuntador de LI una posición después de M.

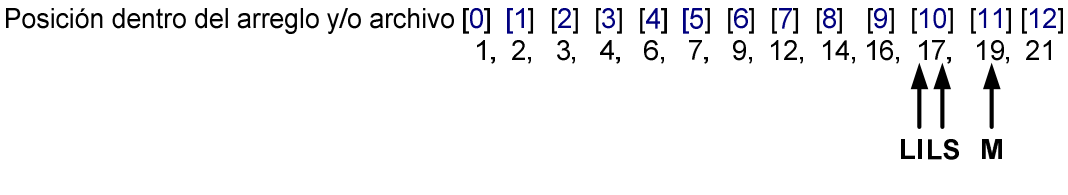


8

Se calcula el valor de M, con los nuevos valores de LI y LS como:

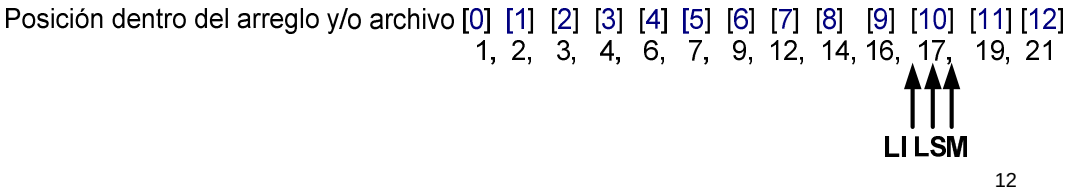
M= Parte Entera ((10 + 12) / 2) = 11

Como el dato (18) es menor que el valor que apunta M (19) se desecha la segunda mitad.



11

Se actualiza la posición de LI, un lugar antes de M.

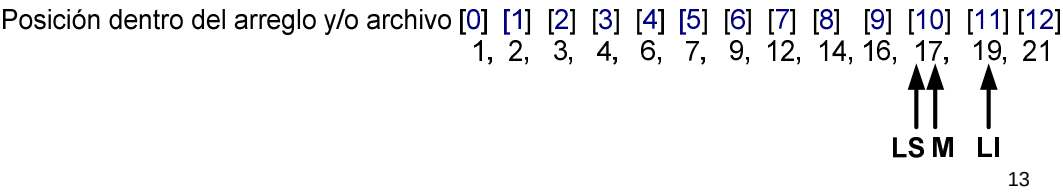


12

Se calcula nuevamente M:

M= Parte Entera ((10 + 10) / 2) = 10

Los tres apuntadores LI, LS y M, apuntan en esta ocasión al mismo elemento que se encuentra en la posición 10 (diez).



Como el dato que se está buscando es 18 (dieciocho), mayor que el valor apuntado por M que es 17 (diecisiete), entonces, se mueve el apuntador LI una posición a la derecha después de M.

Como ya LI no es menor o igual a LM, se asegura que el elemento 18 (dieciocho), que contiene el dato, no se encuentre en la lista.

3.3. Búsqueda mediante transformación de llaves (*hashing*)

La utilización de los métodos por comparación de llaves son muy dependientes del número de elementos, estos es, a mayor número de elementos son mayores las comparaciones.

Existe el grupo de búsquedas por transformación de llaves (Hash), que aumenta la eficiencia, en cuanto al tiempo de ejecución, ya que accede a los registros por lo general más rápidamente, pero va a depender de su implementación.

Entre las ventajas que trae consigo están:

- Aumentar la velocidad de ejecución, independientemente del número de elementos que contenga la lista.
- Los elementos de la lista donde se realiza la búsqueda no forzosamente deben estar clasificados. La lista de datos, puede estar en un archivo o en un arreglo.
- No se necesita recorrer todos los registros anteriores (como en la búsqueda secuencial) o realizar muchas comparaciones (como podría llegar a ser en la búsqueda binaria). Idealmente encontraría directamente el registro que se está buscando.

Entre las desventajas se encuentran:

- Tener que definir un campo llave, que recordando del primer capítulo, es aquel que aparece en todos los registros o nodos de la lista, pero cuyo contenido no se repite.
- La eficiencia de este tipo de búsquedas, va a depender de que tan robusta sea la función HASH, que va a transformar la llave para indicar la posición en la cual se encuentra el elemento buscado, lo ideal es que sea de uno a uno, esto es que para cada valor de llave, esta función la transforme e indique en qué posición va encontrar ese registro.

La transformación de llaves o claves, permite tener acceso a los elementos de manera directa, esto es sin tener la necesidad de recorrer los elementos anteriores, antes de llegar al buscado. Algunos autores también la llaman **función de mapeo**, ya que transforma o mapea la llave de entrada.

Trabaja en base a la función de transformación HASH que comúnmente se denomina como **H (llave),** la cual va a convertir, una llave dada, en una dirección dentro del arreglo o del archivo.

3.3.1. Función Hash

Las funciones de Hash, para lograr su objetivo de proporcionar la dirección donde debe estar el elemento buscado a partir de su llave, lo logran haciendo algunas operaciones sobre los caracteres que la componen.

Es importante hacer notar que la misma función de Hash, que se utiliza, para almacenar, es la misma que se ocupa para recuperar la llave (considerando que sólo se está almacenando por la llave, pero que puede ser un registro o estructura que puede contener varios campos).

Cuando se transforma la llave por el método de suma, por ejemplo al sumar el primer dígito con el tercero, dará como resultado la dirección donde se

almacenará el registro, pero se puede presentar el caso de que la dirección ya esté ocupada por otro elemento, como sería el siguiente caso:

La función de Hash (primer dígito + tercer dígito) aplicado a las siguientes llaves:

K1=123 y K2=292

al aplicar la función de Hash se tendría:

H(K1) = 4

H(K2) = 4

Lo que implicaría que estas dos llaves, con valores diferentes, deberían de estar en la misma dirección, a este suceso se le conoce como **colisión.**

Existen diferentes formas de resolver las colisiones entre las más comunes se tienen las de **direccionamiento abierto** y las de **encadenamiento**.

Para utilizar las búsquedas por transformación de llaves se recomienda considerar las siguientes características:

- Que la función de Hash sea fácil de calcular e implementar.
- También la función de Hash logre la distribución de las llaves lo más uniformemente en la lista (archivo o arreglo).
- Métodos eficientes en para resolver colisiones en caso de que se presenten.

Las funciones de Hash pueden implementarse de acuerdo al tipo de llaves que se van a manejar, por ejemplo, si se va a utilizar el número de folio de las facturas, que emite una compañía dada y estos folios se van a utilizar como llaves, suponiendo que varían del 1 al 99, entonces las llaves convendrían que fueran de tipo numérico y fueran el folio completo.

Pero si el valor de la llave estuviera en un rango, en donde no se ocuparan todos los valores, por ejemplo el número de cuenta de los clientes de tarjetas de crédito de un banco, donde aparecen una gran cantidad de dígitos y pueden que no sean consecutivos, ya que los dos primeros dígitos pueden indicar a lo mejor el número de la sucursal, los otros dos, el tipo de tarjeta (por ejemplo Master Card ®, Visa ®, etc.), los otros dos, a lo mejor el tipo de moneda (pesos, dólares, euros, etc.), entonces no se podría utilizar toda la llave, como función de Hash, ya que habría mucho desperdicio de memoria, por los rangos que no se llegan a utilizar.

El peor de los casos es cuando en la llave se encuentren caracteres que no sólo sean numéricos, como podrían ser también letras como por ejemplo el Registro Federal de Contribuyentes (RFC) o la Clave Única del Registro de Población (CURP), no se podría usar directamente como la dirección donde se almacenara o recuperará, por lo que será necesario realizar algunas conversiones, para manejarla numéricamente, como el asignarle un número a cada letra, ya sea por la posición dentro del abecedario o bien por el valor del código, en el cual se encuentre representado que los más comunes son el ASCII y el EBCDIC.

Entre las transformaciones que se realizan sobre las llaves, se encuentran algunas funciones que pueden involucran desde las operaciones básicas de sumas, restas, multiplicaciones y divisiones, así como las de módulo, cuadrado, cambio de base entre otras. En general las funciones de Hash, el mapear una llave por la dirección, indica en qué dirección debe almacenarse el registros a que pertenece esa llave. Para transformar estas llaves, las funciones de Hash, puede seccionar toda o parte de la llaves, siendo estas partes sobre las que se aplicarán las operaciones antes mencionadas.

Como las operaciones que se utilizan para la transformación puede darse el caso de que den un valor de cero, se debe contemplar la dirección cero en el archivo o en el arreglo si se utiliza un lenguaje de programación que maneje

arreglos con índices a partir de cero, como lo hace **C,** no hay problema, pero si se trata de arreglos como los que utilizan otros lenguajes como FORTRAN, que empiezan en el índice uno será necesario, al resultado de la transformación sumarle uno, para evitar los índices inválidos.

A continuación, se describen algunas de las funciones de Hash más utilizadas.

Función de Hash por suma

Cada parte seccionada de la llave se suma y el resultado puede ser la dirección en la cual se almacenará o recuperará el elemento. Estas partes seccionadas pueden ser todos los dígitos de la llave o sólo algunos de ellos y el resultado se toma completo con los dígitos que dé o también sólo algunos de ellos.

Es importante que cuando se implemente cualquier función de Hash primero se analice, para considerar que espacio de memoria para la lista se le va a asignar, por ejemplo si se toma este caso de la suma y que las llaves consisten de tres dígitos, entonces, el rango de valores que pueden tener es del 0 al 999, o sea 1000 localidades, pero al sumarse todos los dígito el rango sería del 0 al 9 + 9 +9 = 27, esto es solo 28 posiciones.

Es importante resaltar, que la suma puede involucrar todos los dígitos o sólo algunos de ellos.

Ejemplo

Supóngase que se tienen las siguientes llaves (K):

K1 = 125

K2 = 239

K3 = 005

La función de Hash indica que la transformación es la suma de todos los dígitos que componen la llave como a continuación se muestra:

H (K) = digito 1 + digito 2 + digito 3

Al aplicar la función de Hash a cada llave las salidas son:

H (K1) = H (125) = 1 + 2 + 5 = 8

H (K2) = H (239) = 2 + 3 + 9 = 14

H (K1) = H (005) = 0 + 0 + 5 = 5

El registro cuya llave es 125 debe de estar en la dirección 8, el de la llave 239 en la 14 y para la 005 en la 5.

Función de HASH por multiplicación

Existen diferentes alternativas, en la multiplicación, ya que pueden multiplicarse el valor de toda la llave por sí misma, por una constante, separar por dígitos, realizar la multiplicación de determinados dígitos por otros que componen a la llave, etc. y nuevamente el resultado se toma completo con los dígitos que se obtengan o sólo un grupo de ellos.

Cuando se multiplica por sí mismo el valor de la llave se conoce como la operación cuadrado y se ha observado que, para evitar o reducir las **colisiones**, se utilizan los dígitos centrales de la operación.

Ejemplo

Se tienen las siguientes llaves (**K):**

K1 = 125

K2 = 239

K3 = 005

La función de Hash que se utilizará, indica que la transformación es la multiplicación del valor de la llave por sí misma y que se tomaran los primeros tres dígitos del resultado, para obtener la dirección del registro a donde pertenece esa llave.

H (K) = tres primeros dígito de (K * K)

Al aplicar la función de Hash a cada llave, las salidas son:

H (K1) = H (125) = 125 * 125 = 15625 = 156
H (K2) = H (239) = 239 * 239 = 57121 = 571
H (K1) = H (005) = 5 * 5 = 025

El registro cuya llave es 125 debe estar en la dirección 156, el de la llave 239 en la 571 y para la 005 en la 25.

Ejemplo
En este ejercicio la función de Hash indica que la transformación es la multiplicación del valor de la llave por una constante que se supondrá con el valor de 17 y que se tomaran los últimos tres dígitos del resultado, para obtener la dirección del registro a donde pertenece esa llave.

H (K) = tres últimos dígito de (K * 17)

Al aplicar la función de Hash a cada llave, las salidas son:

H (K1) = H (125) = 125 * 17 = 2125 = 125
H (K2) = H (239) = 239 * 17 = 4063 = 063
H (K1) = H (005) = 5 * 17 = 085

Ahora, el registro cuya llave es 125, debe estar en la dirección 125, el de la llave 239 en la 063 y para la 005 en la 085.

NOTA: Hay que tener en cuenta que cuando se separan los dígitos de la llave y se multiplican entre una combinación de ellos puede darse el caso de que aparezcan muchos valores en ceros.

Por ejemplo, si se multiplicara el primero por el tercer dígito y las llaves están en el rango de 0 a 999, entonces los primeros cien elementos, van a dar cero,

aparte de los que terminen sus llaves en cero o su segundo dígito también fuera cero.

Función de Hash por la operación módulo

Va muy relacionado con la división, sólo que en este caso en particular, se toma el residuo, como resultado de la función de Hash y nuevamente se puede tomar el resultado completo con todos los dígitos que del resultado o solo un grupo de ellos.

Se sugiere para evitar o reducir las **colisiones** y para que se distribuyan más uniformemente los elementos, que el divisor sea un número primo cercano a la longitud de elementos del arreglo o del archivo, según sea el caso.

Es importante resaltar que el residuo va a variar entre cero y el valor del divisor menos uno, pero que finalmente se tendrán un rango de valores igual al valor del divisor.

Ejemplo

Se tienen las siguientes llaves (**K):**

K1 = 125

K2 = 239

K3 = 005

La función de Hash indica que la transformación es el módulo con todos sus dígitos del valor de la llave entre N, que para este ejemplo se tomó N = 20, por lo cual se tendrán resultados entre cero y diecinueve (veinte valores).

$H(K) = K \bmod N$

Al aplicar la función de Hash, a cada llave las salidas son:

$$H(K1) = H(125) = 125 \bmod 20 = 05$$

$$H(K2) = H(239) = 239 \bmod 20 = 19$$

$$H (K1) = H (005) = \quad 5 \bmod \quad 5 = 05$$

El registro cuya llave es 125, debe estar en la dirección 5, el de la llave 239 en la 19 y para la 005 en la 5.

Se observa que apareció con esta función de Hash, una colisión, ya que para las llaves 125 y para la 5, indica que deben estar las dos en la dirección **5**, pero en una dirección solo puede estar un elemento, no más.

Lo ideal es que no se presenten, pero si se dan, no hay que preocuparse, más adelante se tratarán algunas de las formas para la solución de las colisiones.

Así como se tomó el residuo, para la función módulo, también se pudo haber empleado la división y tomar la parte entera del cociente, en todos sus dígitos o algunos de ellos.

Función de Hash por la operación cambio de base

Esta función consiste en cambiar la llave que se encuentra en una base numérica dada, por lo general base diez a otra base cualquiera, se ha observado empíricamente, que se reduce considerablemente el número de **colisiones,** cuando se cambia a una base más pequeña, por ejemplo de **base diez** a base **cuatro.**

Ejemplo:

Se tienen las siguientes llaves **(K):**

K1 = 125

K2 = 239

K3 = 005

La función de Hash indica que la transformación es el cambio de base con todos sus dígitos del valor de la llave que se encuentra en base diez y se va a cambiar a base cuatro.

H (K) = cambiar de baseOrigen a baseDestino a K
H (K) = cambiar de base 10 a base 4 a K

al aplicar la función de Hash a cada llave, las salidas son:

H (K1) = H (125) = 1331
H (K2) = H (239) = 3233
H (K1) = H (005) = 0011

El registro cuya llave es 125, debe estar en la dirección 1331, el de la llave 239 en la 3233 y para la 005 en la 11.

Función de Hash por plegamiento

Se basa en dividir la llave en grupos iguales de dígito, aunque puede darse el caso que dependiendo del número de dígitos de las llaves, el último grupo puede ser de diferente tamaño a los otros. Una vez teniendo los grupos, manipularlas con las operaciones básicas.

Por ejemplo, se aplica la función de Hash de suma tomando subgrupos de dígitos que componen la llave principal y del resultado tomar algunos o todos los dígitos, sea el caso de contar con cinco dígitos de la llave, se pueden tomar dos subgrupos, uno de dos y otro de tres dígitos y posteriormente sumar las cantidades, que se lean de cada uno de esos subgrupos (tomando en cuenta todos los dígitos de ese subgrupo, si se tiene la cantidad de 12 en el primer subgrupo de dos dígitos y de 245 en el segundo subgrupo de tres dígitos, el resultado será de 12 + 245 = 257).

Generalmente se ha visto que se reducen las **colisiones,** cuando se toman los dígitos menos significativos, ya que son los que varían mucho más rápido, que los más significativos. Se pueden tomar grupos de dígitos, inclusive de uno solo y pueden ser grupos de diferentes longitudes.
Los dígitos se tomarán como a continuación se muestra:

$K = k_n \ldots\ldots\ldots k_2 k_1 k_0$

Ejemplo

Se tienen las siguientes llaves (**K):**

K1 = 125

K2 = 239

K3 = 005

La función de Hash indica que la transformación es agrupar conjuntos de dos y otro de uno elemento, aplicar la operación suma y del resultado tomar sólo los dos últimos dígitos.

H (K) = dos dígitos menos significativos de la suma de $(d_2\, d_1 + d_0)$

al aplicar la función de Hash, a cada llave las salidas son:

H (K1) = H (125) = 12 +5 = 17

H (K2) = H (239) = 23 +9 = 32

H (K1) = H (005) = 00 +5 = 05

el elemento de la llave 125, debe estar en la dirección 17, el de la llave 239 en la 32 y para el 005 en la 05.

Función de Hash por truncamiento

Consiste en la separación de cada uno de los dígitos que componen a la llave y tomar algunos de ellos para formar la dirección que sería la salida de la función de Hash.

Es uno de los métodos más sencillos, pero como los dígitos pueden venir aleatoriamente, la distribución a veces no es tan uniforme y pueden llegarse a presentar una gran cantidad de colisiones.

Ejemplo

Se tienen las siguientes llaves (K):

K1 = 125

K2 = 239

K3 = 005

La función de Hash indica que la transformación es agrupar conjuntos de dos y otro de uno elemento, aplicar la operación suma y del resultado tomar sólo los dos últimos dígitos.

H (K) = dos dígitos menos significativos de la posición par de (d_2 d_0)

Al aplicar la función de Hash a cada llave, las salidas son:

H (K1) = H (125) = 1 + 5 = 06

H (K2) = H (239) = 2 + 9 = 11

H (K1) = H (005) = 0 + 5 = 05

El elemento de la llave 125, debe estar en la dirección 06, el de la llave 239 en la 11 y para el 005 en la 05.

Como se observa, se utilizaron las mismas llaves en todos los ejemplos y en algunas funciones de Hash la dirección que devuelve se llegó a repetir, como es el caso de la llave igual a cinco pero, generalmente fueron direcciones diferentes y en el caso de la función de **módulo** existió una **colisión**, en la dirección cinco.

Esto demuestra que no hay alguna regla que indique cual es la mejor forma de realizar la función de mapeo o de Hash y que inclusive se pueden realizar con combinaciones de las diferentes funciones que se presentaron o tomar una mayor o menor cantidad de dígitos de los resultados.

Será necesario, antes de implementarla, realizar pruebas con alguna muestra representativa de las llaves que se vayan a manejar y ver si conviene en cuanto a la cantidad de **colisiones**, que se presentan.

Para realizar lo más eficientemente posible la función de mapeo, existen algunos conceptos de la Probabilidad y Estadística, que pueden ser de gran ayuda, como por ejemplo para el análisis de la distribución de los datos y los criterios para el tamaño y toma de la muestra.
Sin embargo no se asegura la eliminación de las colisiones.

Análisis de los algoritmos de Búsquedas

En el caso de búsquedas por comparación de llaves se pueden utilizar, cuando el número de elementos de la tabla es relativamente pequeño, ya que pierde rapidez al crecer la tabla. Sin embargo, tiene la facilidad de sus algoritmos y los elementos de la tabla pueden estar almacenados en forma contigua o ligada.

Para las búsquedas por transformación de llaves se sugiere que para aumentar la eficiencia de los algoritmos de búsqueda por transformación de llaves se le asigne a la tabla un número de localidades de almacenamiento mayor al número elementos que se desean almacenar. Se puede establecer la proporción de la tabla que está ocupada por un factor de carga expresado como:

α = Numero de nodos a insertar / Número de nodos para almacenamiento[19]

En muy buena medida para que al algoritmo de este tipo de búsquedas tenga una mayor eficiencia, va a depender de la función de mapeo o de Hash y de la forma de resolver las colisiones. Idealmente se desea que no se presenten colisiones, pero no siempre se logra, en caso de presentarse, se busca que sean las menos posibles y resolverlas con una técnica eficiente.

3.3.2. Resolución de colisiones

[19] *Ibidem*, p. 192.

Para poder resolver el problema de las colisiones se busca alguna dirección que se encuentre desocupada cercana a la posición en donde debe de estar el nodo o la estructura (de acuerdo a la que proporcione la función de Hash).

Direccionamiento Abierto

Consiste en revisar las direcciones de la tabla (así se le puede llamar al archivo o al arreglo donde se esté almacenando y/o recuperando las llaves procesadas) en forma secuencial, partiendo de la dirección donde ocurrió la colisión (misma dirección para más de una llave diferente), hasta encontrar el elemento deseado si se trata de una búsqueda o bien encontrar un espacio libre si es el caso de almacenamiento.

Cuando se trata de almacenar un elemento se asigna la primera localidad disponible más cercana al nodo y si se trata de una búsqueda para consulta para realizar **a**ltas, **b**ajas y **c**ambios, se busca en forma secuencial a partir de esa posición, hasta encontrar el elemento buscado, pero si se encuentra una dirección disponible sin haber localizado la llave del nodo deseado, indicaría que el elemento no se encuentra en la tabla o archivo.

Tal vez sea la forma más sencilla para resolver las colisiones, pero tiene la desventaja de que si se presentan varias colisiones, entonces se empiezan a agrupar nodos o elementos alrededor de una misma dirección (***clustering***), en forma contigua y al realizar otra consulta, esta se puede llegar a convertir en una búsqueda secuencial o lineal, llevando esto el consumo de más tiempo de ejecución del algoritmo y con el consiguiente aumento en el tiempo de respuesta de la consulta.

Para encontrar la dirección disponible se recorre la tabla a partir del punto donde ocurrió la colisión hacia abajo y si se llega hasta el final de la tabla, entonces se regresa a buscar a partir de la primera dirección, como si se tratara de una cola circular.

En el peor de los casos, si se presentaran muchas colisiones, es posible que se le tenga que dar toda una vuelta a la tabla para encontrar el nodo deseado.

Por eso la importancia de dejar aproximadamente un 20 % más de memoria que la que se estima ocupar, por ejemplo si se van a almacenar 1,000 elementos, se recomienda reservar 1,200 localidades para que existan direcciones desocupadas, distribuidas en ese espacio de memoria, para evitar tener que darle la vuelta completa a la tabla(puede ser archivos o arreglos).

Encadenamiento

En esta forma de solución de colisiones se crea una lista simplemente ligada, para cada dirección que se repita, esta lista ligada, indicará cual es la dirección del siguiente elemento que debió de haber estado en la localidad donde se esté originando la colisión, con esto ya no se realiza una búsqueda secuencial, elemento por elemento, si no que se recorre la lista ligada a través de las direcciones de las ligas hasta que encuentre la dirección NULL o Nulo, logrando una mayor velocidad de esta técnica. Pero tiene la desventaja de que ocupa un espacio de memoria mayor para almacenar la liga y que el algoritmo se vuelve un poco más complejo comparado con el de Direccionamiento Abierto.

3.4. Árboles binarios de búsqueda

Otra de las herramientas básicas para realizar búsquedas, es a través de los árboles binarios de búsqueda, para lo cual se recordarán algunos conceptos, como la definición de árboles y de grafos, a continuación se explican detalladamente:

Grafo o gráfica o diagrama

La forma más conocida de representar un grafo es por medio de un dibujo donde se tienen los siguientes elementos:

- Los puntos, también llamados vértices, nodos, o unión.
- Las líneas que unen a los puntos, conocidas como arcos, ramas o elementos.

Para representar en la memoria de una computadora al grafo **G= (E, V)** será necesario utilizar dos conjuntos de elementos;
uno para los arcos

E = (e1, e2,....en)

y otro para los vértices

V = (v1,v2,....vn)

Por ejemplo:

V = (v1, v2, v3, v4, v5, v6, v7, v8)
E = (e1, e2, e3, e4, e5, e6, e7, e8)

Árboles.

Un **árbol** es un grafo que tiene las siguientes **características**:

- Es un grafo simple y finito, ya que no tiene ni ciclos, ni arcos paralelos y tiene un conjunto finito de nodos.
- Hay un nodo específico llamado raíz y es el único que no tiene ningún nodo del cual descienda, esto es, el nodo del cual descienden todos los demás que componen al árbol.
- El resto de los nodos están separados en n conjuntos desordenados, denominados T_1 , T_2 , T_3, . . . T_n donde n es mayor o igual a cero. Cada uno de estos conjuntos son a su vez árboles, llamados subárboles de la raíz. Esta parte de la definición es recursiva.
- Existe sólo un patrón entre cada par de vértices. Un patrón es un conjunto de arcos y nodos, que parten de un origen y llegan a otro nodo destino, en el cual ni vértices ni arcos aparecen más de una vez.

- El número de vértices n que contiene es igual a n-1 arcos.
- La suma de los grados de todos los vértices es 2(n-1), donde n es el número de vértices del grafo.

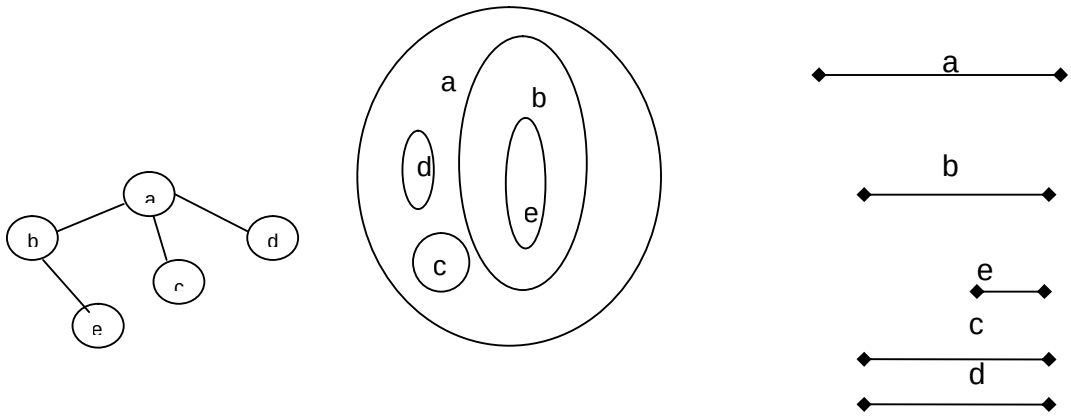
Niveles del árbol

Un árbol es como un árbol genealógico en donde a cada generación se le conoce como nivel. De todos los nodos que componen al árbol, existe uno que no tiene antecesor, que se llama **raíz**, el cual para algunos autores es el nivel cero y para otros el nivel uno, también vale la pena mencionar que el nivel de un nodo es igual al nivel de su antecesor más uno.

Los nodos que tienen descendientes se les conocen como **nodos padres** y a los descendientes como **nodos hijos.** Un nodo padre puede tener 0 (cero) o varios descendientes, pero un nodo descendiente no puede tener más que a un solo nodo padre. No existe ninguna relación entre los nodos del mismo nivel, solo hacia sus descendientes o ascendentes.

Representación de árboles

Un árbol se puede representar en forma gráfica ya sea en forma de barras o de conjuntos.

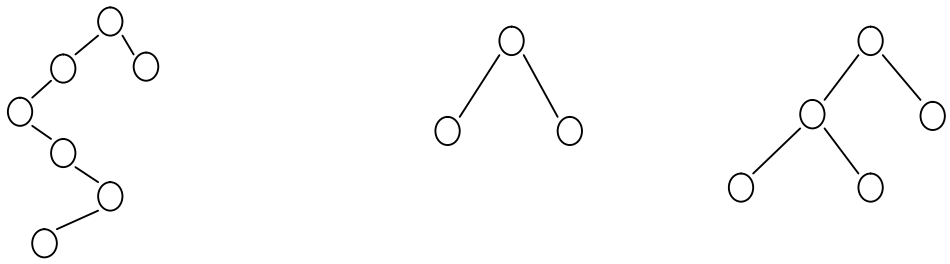


GRAFICA CONJUNTOS TABULAR

Figura 3.2. Representación de árboles

Árbol de Knuth y estrictamente binario

Un árbol es binario o de Knuth cuando puede tener cero, uno dos descendientes, y si sólo puede tener cero o dos descendientes se llama estrictamente binario.



ÁRBOL BINARIO ÁRBOLES ESTRICTAMENTE BINARIOS

Figura 3.3. Árbol de Knuth y estrictamente binario

Árboles binarios

Tienen varias aplicaciones como:

- Implementación de diferentes algoritmos como el de clasificación del Heap, visto con anterioridad.
- Aplicaciones de búsqueda binaria.

Para los árboles estrictamente binarios y binarios, se pueden ocupar los cinco tipos de **recorridos clásicos**, que son los que describen los siguientes algoritmos:

- Arriba-abajo.- el recorrido es partiendo de la raíz hacia abajo y de izquierda a derecha.
- Abajo-arriba.- parte del nivel más bajo hacia arriba y de izquierda a derecha.
- Preorden.- visitar raíz, recorrer rama izquierda y recorrer rama derecha.
- Inorden.- recorrer rama izquierda, visitar raíz y recorrer rama derecha.
- Posorden.- recorrer rama izquierda, recorrer rama derecha y visitar raíz.

Estos recorridos son utilizados en algunas de las aplicaciones anteriormente mencionadas.

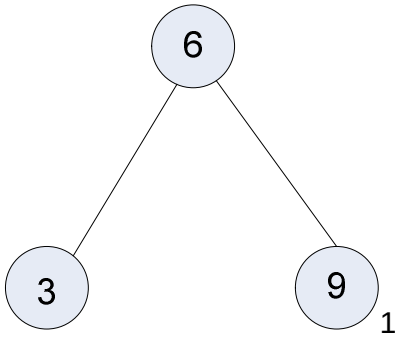
Cuando el algoritmo indica que se visita la raíz es cuando se recupera el dato de ese nodo y cuando se menciona que se recorra alguna de las ramas izquierda o derecha, se realiza, siempre y cuando existan esas ramas del árbol.

Árboles de búsqueda binaria (ABB)

Es un árbol binario que cumple con las siguientes características:

- El dato que contiene cada uno de sus nodos no debe de estar repetido.
- El valor del dato que contiene la raíz de la rama izquierda (si es que existe), debe de tener un valor menor al de la raíz original.

- El dato de la rama derecha es mayor al de la raíz.

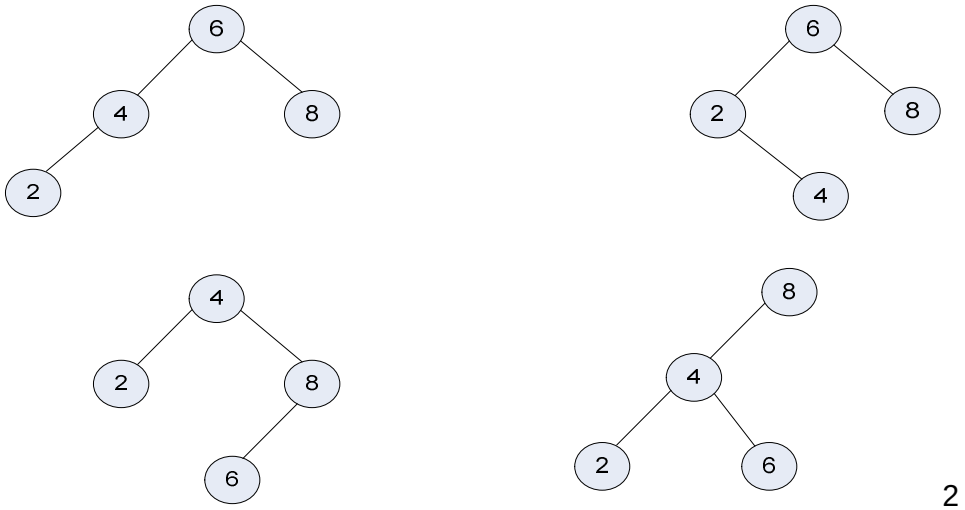


1

Figura 3.4. Árboles de búsqueda binaria (ABB)

Como se observa la raíz del árbol, es un valor mayor al de la raíz de su rama izquierda, pero menor al valor de la raíz de la rama derecha, esta característica es la que se aprovecha en las búsquedas.

El árbol binario de búsqueda para un conjunto de elementos puede representarse de diferentes formas, como se muestra en el siguiente ejemplo.



2

Figura 3.5.Diferentes representaciones de un árbol binario de búsqueda

El recorrido Inorden de cualquier de estos árboles binarios de búsqueda, produce la misma secuencia clasificada en forma ascendente.

Operaciones

Existen diferentes operaciones que se pueden realizar en este tipo de árboles, entre las que se encuentran:

- Búsquedas.
- Inserciones.
- Bajas
- Búsquedas

Los pasos a seguir son:

```
inicio
   si el árbol esta vacio entonces
         desplegar “árbol vacio y no se encuentra el elemento”
   en caso contrario
        si el valor del dato del nodo raíz es igual al del elemento que se busca
            desplegar “búsqueda con éxito”
        en caso contrario
            si  el valor del dato buscado es < al valor del nodo raíz
                  se continúa la búsqueda en el subárbol izquierdo
                 (llamándose recursivamente este algoritmo)
            en caso contrario
                  si el valor del dato que se desea encontrar es > que el valor
                  del nodo raíz entonces
                        búsqueda se realizará en el subárbol derecho
```

SI el elemento buscado se encuentra en alguna dirección se puede dar a través de un apuntador que la indique, pero si no lo encuentra entonces se tendrá la dirección de NULL (Nulo), en dicho apuntador.

Ejemplo

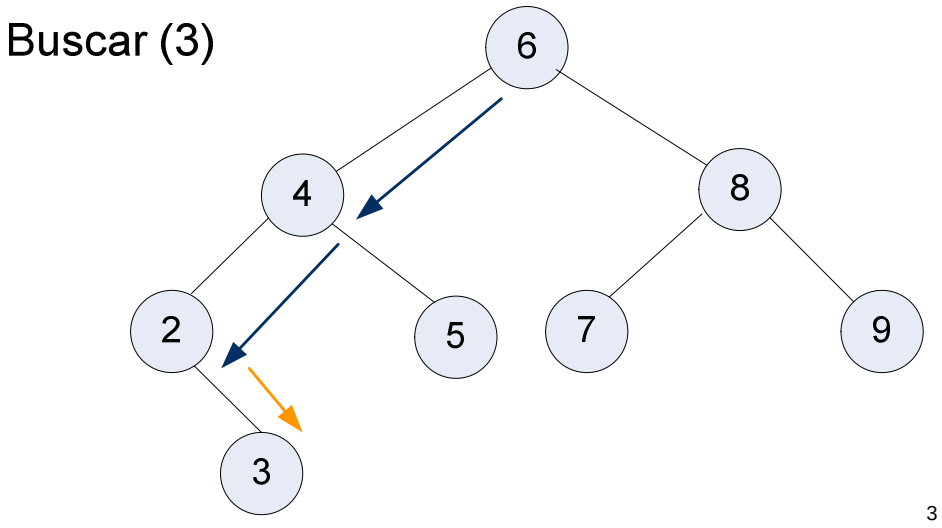


Figura 3.6. Búsqueda en un Árbol Binario de Búsqueda

Para buscar el elemento 3, se recorrió el árbol a partir de la raíz dos veces a la izquierda, ya que 3 es menor que 6, después 3 es menor que 4 (flechas color verde –azul oscuro) y después a la derecha, ya que 3 fue mayor que 2 (flecha color anaranjado claro).

Inserciones

Para insertar un elemento se utiliza el algoritmo de búsqueda, ya que si el elemento no está en el árbol, es insertado a continuación del último nodo visitado, en caso contrario no se realizará acción alguna.

Se requiere de un apuntador auxiliar, para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese apuntador es NULL.

A continuación se describe en pseudocódigo el algoritmo a seguir para insertar un nodo:

```
inicio
padre = NULL
nodo = dirección de la raíz
mientras el nodo actual sea diferente de NULL (no sea un árbol vacío) o hasta que se encuentre el elemento.
si el valor del nodo  es > que el elemento buscado // nodo esta
                    //apuntando a la raíz actual
          padre = nodo          //continua la búsqueda en el árbol izquierdo
          nodo =Liga Izquierda ( nodo )
en caso contrario
      padre = nodo          //continua la búsqueda en el árbol derecho
      nodo=Liga Derecha (nodo)
fin
fin
si nodo es diferente de NULL
          el elemento existe en el árbol
          desplegar “no se inserta el elemento, ya existe”
en caso contrario
          si padre = NULL    //el árbol estaba vacío, por lo tanto, el nuevo árbol
                             //sólo contendrá el nuevo elemento, que será la raíz
                             //del árbol
               insertar dato en la raíz principal del árbol
          en caso contrario
             si  elemento <  padre   //entonces se inserta el nuevo elemento como
                                     //un nuevo árbol izquierdo del padre
                  insertar dato como nuevo árbol izquierdo del padre
```

```
            en caso contrario
                                    //entonces se inserta el nuevo elemento
                                     //como un nuevo árbol derecho del padre
                    insertar dato como nuevo árbol derecho del padre
              fin
          fin
     fin
fin
```

Bajas

Para borrar un elemento o nodo también se necesita de la función de búsqueda, ya que si el elemento no está en el árbol no se puede borrar.

Hay dos alternativas:

1. Se trata de un nodo hoja (aquel nodo que ya no tiene descendientes), se puede borrar directamente.

2.. En caso de tratarse de un nodo que sea una rama, no se puede eliminar directamente ya que se perderían todos los elementos de ese subárbol.

Se procede a buscar el nodo más a la izquierda del subárbol derecho, o el que está más a la derecha del subárbol izquierdo y se intercambian sus valores. Procediéndose a eliminar el nodo hoja.

.

Nuevamente se hace uso de un apuntador auxiliar para conservar una referencia o dirección al padre del nodo raíz actual y se inicializa con un valor de NULL (Nulo).

Debido a que este algoritmo es un poco más complejo explicarlo con un pseudocódigo, se va a proceder a realizarlo con dos ejemplos, el primero de ellos, se va a borrar un nodo terminal u hoja (grado del nodo igual a uno) y otro ejemplo donde el nodo que se va a eliminar será uno intermedio.

Para clarificar la explicación, se va a considerar que cada nodo, solo va a tener tres campos; uno de **dato,** otro de la **Liga Izquierda** que apunta a la rama

izquierda y otro que será la **Liga Derecha**, que apuntará a la rama derecha de dicho nodo.

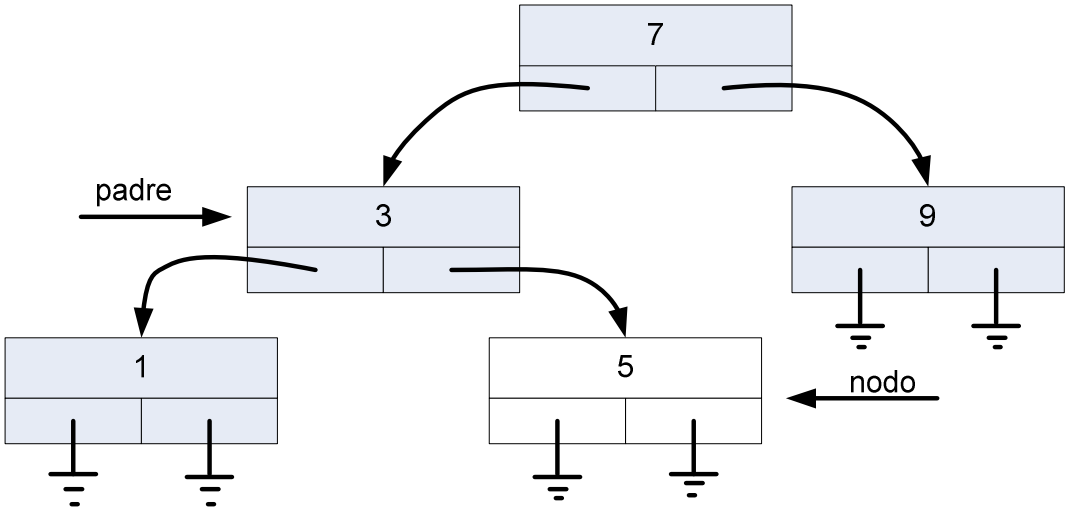
Estas ligas, que en realidad son apuntadores dentro de cada nodo, pueden tener un valor de NULL, cuando no exista alguna o las dos ramas como a continuación se ejemplifica en la ilustración de un nodo.

4

Ejemplo de borrado de nodo terminal

Se desea eliminar un nodo que es una hoja, para este caso se desea eliminar el nodo 5. Para lo que se procede a realizar los siguientes pasos:

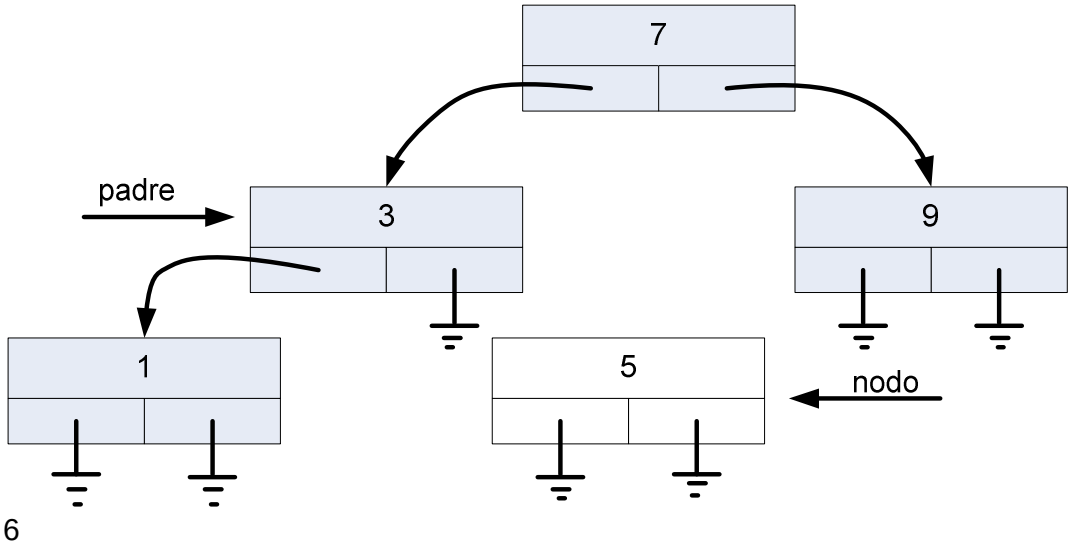
Localización del nodo a borrar, a través del algoritmo de búsqueda, visto anteriormente y se conserva el apuntador al nodo padre.



5

Figura 3.7. Borrado de un nodo terminal, localización del nodo

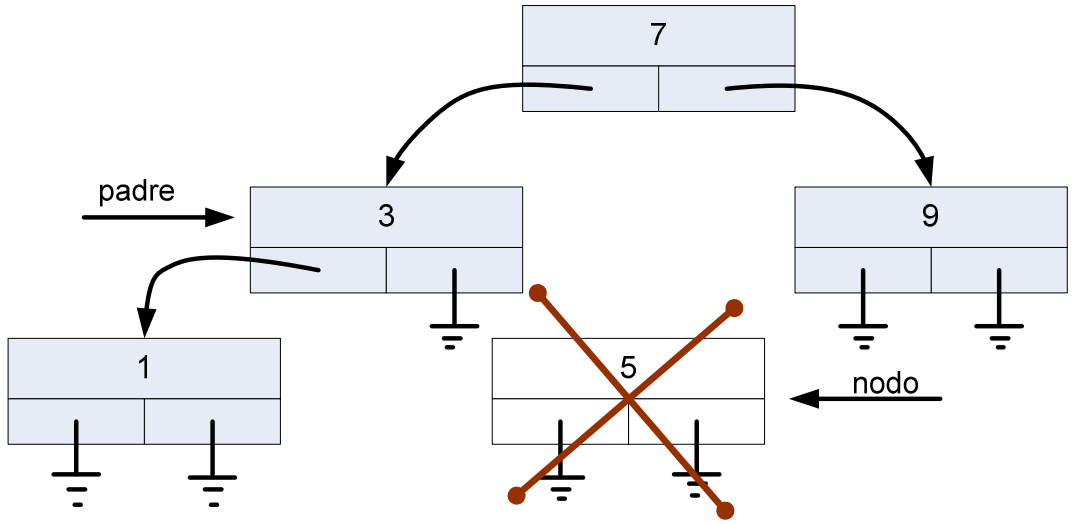
La Liga Izquierda o Derecha del nodo padre que apuntaba a nodo, ahora apunte a NULL. En este ejemplo fue la liga derecha.



6

Figura 3.8. Actualización de la liga

Se elimina el nodo.



7

Figura 3.9. Eliminación del nodo

Ejemplo de borrado de nodo rama con intercambio de un nodo hoja

Para este caso se hace un intercambio con un nodo hoja, en este ejemplo se desea eliminar el nodo **7**, para lo cual se procederá a realizar los siguientes pasos:

1.- Se localiza el nodo a borrar y es apuntador por **raíz**.

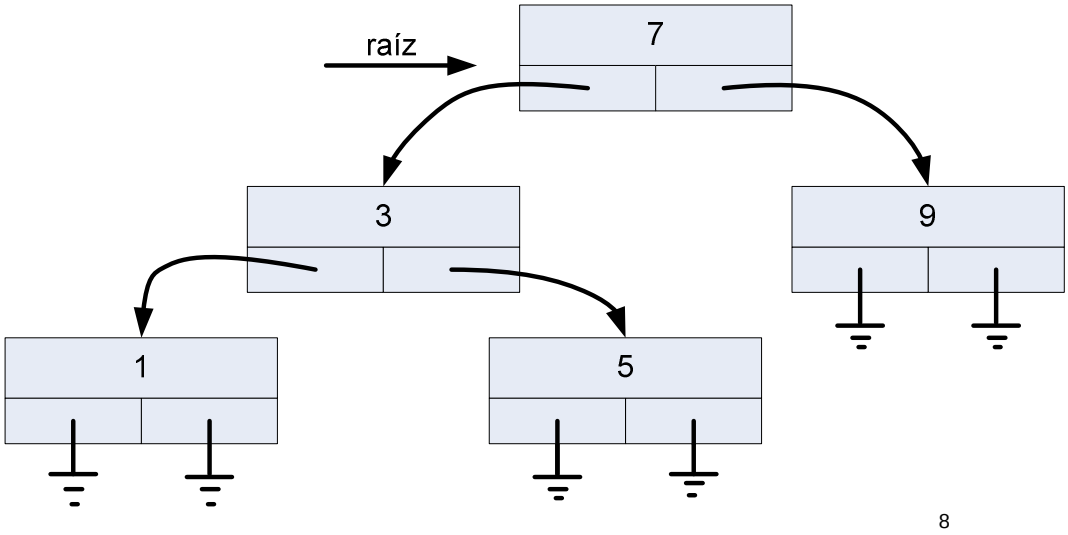


Figura 3.10. Localización del nodo no terminal, para su borrado

2.- Se busca el nodo más a la derecha del árbol izquierdo de raíz, en este caso el 5 y ahí queda el apuntador de **nodo**, se conserva un apuntador al nodo padre de nodo, al cual se denomina **padre**.

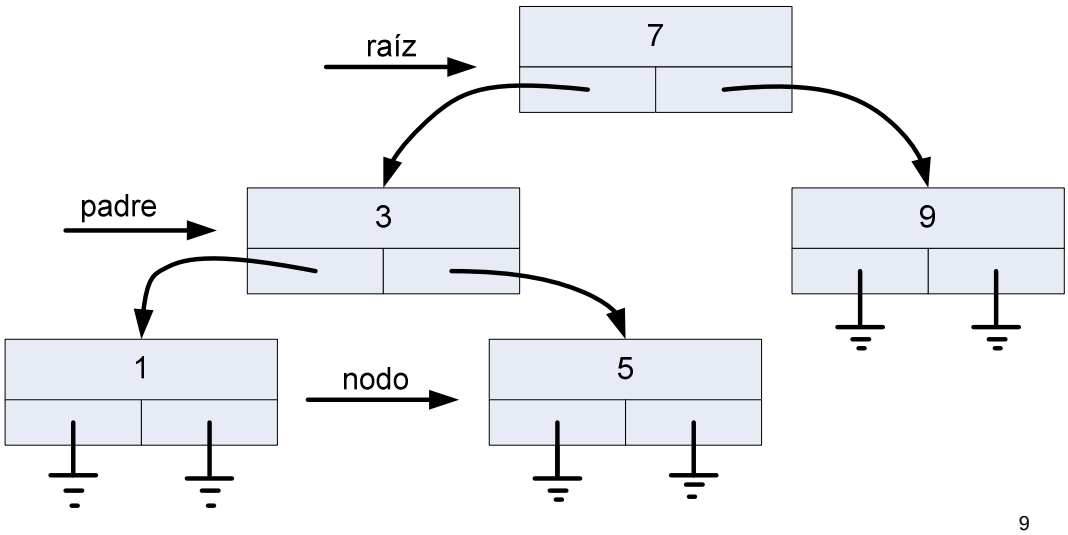
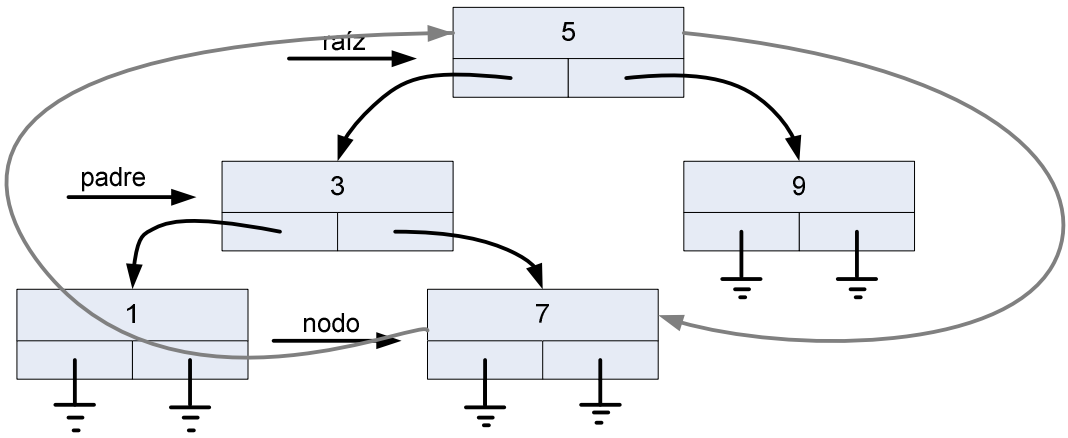


Figura 3.11. Inicailización de apuntadores

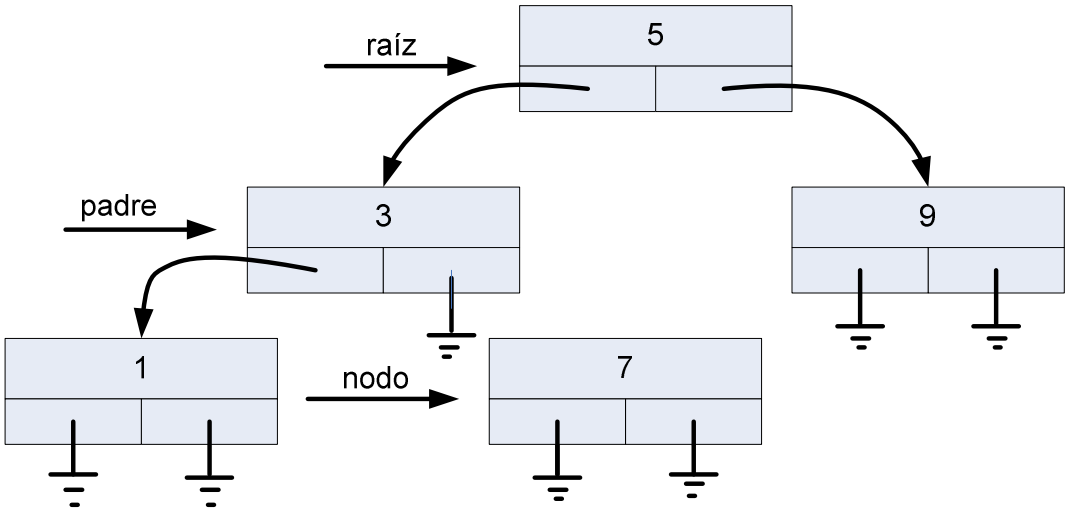
3.- Se intercambia el dato de **nodo** por el de **raíz**



10

Figura 3.12. Intercambio del dato de nodo por el de raíz

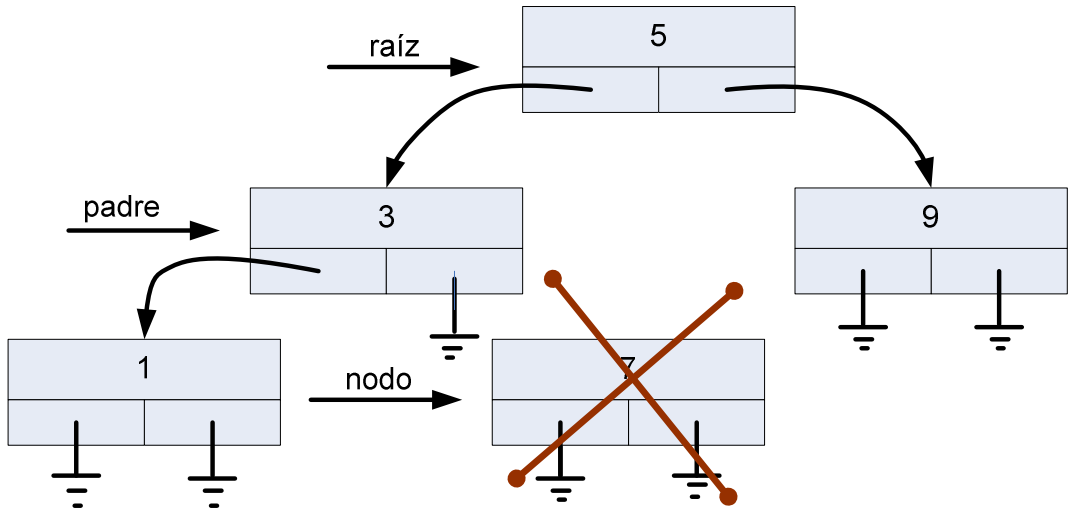
4.- La liga Derecha de padre, va a apuntar a NULL.



11

Figura 3.13 . Actualización de la liga derecha del padre

5.- Se borra el nodo



12

Figura 3.14. Borrado del nodo

Los conceptos de este tema, se pueden resumir en la importancia, que tienen los métodos de búsqueda, así como su estrecha vinculación con los métodos de clasificación.

A pesar de que han aparecido nuevas tecnologías, tanto en hardware, como en software, siguen siendo una de las bases para el desarrollo de sistemas de información.

Sobre todo, es importante conocer estos conceptos de los tres temas de este apunte, porque aún utilizando herramientas de tecnología de punta, como son los manejadores de Bases de Datos o lenguajes de cuarta generación (4GL), etc., el contar con los conocimientos de archivos, búsquedas y clasificación (ordenamientos), ayudan a aprovechar mejor el uso de estas herramientas modernas.

Para finalizar se cierra con la siguiente frase de Ackoff que dice: “se ha observado que es más frecuente no enfocar el problema correcto, que fallar en la solución del problema que se enfoca”.[3,] por lo que es importante contar con herramientas que ayuden en la solución del problema como las que se mencionan en este trabajo, en cuanto a algoritmos y Estructuras de Datos se refiere, pero también es importante desarrollar las habilidades, para hacer un buen planteamiento y formulación del problema.

Bibliografía del tema 3

EUÁN Ávila, Jorge Iván y CORDERO Borboa, Luis Gonzaga, *Estructuras de datos*, México, Limusa, 1989, pp. 219.

JOYANES Aguilar, Luis, *et. al*, *Estructuras de datos en C*, España, Mc Graw Hill, 2005, pp. 435.

ROSE Gómez, César E., *Archivos, Organización y Procedimientos,* México, Computec**,** 1993, pp. 227.

Actividades de aprendizaje

A.3.1 Elabora un mapa conceptual de este tercer tema.

A.3.2 Realiza un programa en el lenguaje de programación C para implementar la búsqueda binaria, partiendo del pseudo código visto en este tema

A.3.3 Implementa un programa C para desarrollar la búsqueda en un árbol de búsqueda binaria, utilizando la sentencia *struct*.

A.3.4 Explica cómo se clasifican los métodos de búsqueda.

A.3.5 Describe en qué consiste cada uno de los métodos utilizados para resolver las colisiones.

---

[3] Ackoff Rusesell L. Ackoff, *El arte de resolver problemas,* pp.25-26.

Cuestionario de autoevaluación

1. ¿Qué es una colisión?
2. ¿Qué es una función de Hash?
3. ¿Con qué métodos se resuelven las colisiones?
4. ¿Cómo se define un Árbol de Búsqueda Binaria?
5. ¿Cómo se llama el recorrido, que se utiliza en un Árbol de Búsqueda Binaria, que siempre muestra los nodos en forma ascendente?
6. ¿Los métodos de búsqueda se dividen en?
7. ¿Cómo se llaman los métodos de búsqueda por comparación de llaves?
8. ¿Qué método de búsqueda, es independiente del número de elementos, sobre los que va a actuar?
9. En la función de Hash por la operación cambio de base, ¿cuál es la recomendación, para reducir el número de colisiones**,** en cuanto a el manejo del tamaño de las bases?
10. ¿Cuál es el método de búsqueda que requiere que los elementos, estén previamente clasificado, en forma ascendente o descendente?

Examen de autoevaluación

Relaciona la columna de la izquierda con la de la derecha y coloca la respuesta que consideres correcta dentro del paréntesis.

( ) 1. Árbol cuya rama izquierda tiene un valor menor y la rama derecha mayor al de su raíz	a) Árbol binario
( ) 2. Son los elementos de un Grafo	b) Encadenamiento
( ) 3. Es un grafo que sólo puede tener cero, uno o dos descendientes	c) Búsqueda binaria.
( ) 4. Es la acción de localizar un nodo o elemento en particular dentro de una lista (archivo o arreglo).	d) Colisión
( ) 5. Técnica para resolver las colisiones, que revisa la tabla en forma secuencial, hasta encontrar el elemento deseado o un espacio libre	e) Búsqueda por comparación de llaves lineal
( ) 6. Técnica para resolver las Colisiones, que utiliza una lista simplemente ligada	f) Árboles.
( ) 7. Es una búsqueda, donde es necesario que las llaves estén clasificadas ascendente o descendentemente	g) Árbol binario de búsqueda
( ) 8. Este tipo de búsqueda, puede o no tener las llaves clasificadas	h) Direccionamiento abierto
( ) 9. Ocurre cuando al utilizar una función de Hash, esta le asigna a un nodo una dirección que ya ha sido ocupado por otro diferente	i) Búsqueda
( ) 10. Las representaciones gráfica, de conjuntos y tabular sirven para representar	j) Punto y líneas.

BIBLIOGRAFÍA BASICA:

AHO, Alfred V., HOPCROFT, John E. y ULLMAN, Jeffrey D. ULLMAN, *Estructuras de Datos y Algoritmos*, Delaware USA, editorial Addison-Wesley Iberoamericana, 1988, pp. 438.

BOWMAN, Charles F., *Algoritmos y Estructuras de Datos Aproximación en C*, México, Oxford University Press, 1999, pp. 333.

EUÁN Avila. Jorge Iván y CORDERO BORBOA, Luis Gonzaga, *Estructuras de datos*, México, Limusa, 1989, pp. 219.

FRANCH Gutiérrez, Xavier., *Estructuras de datos Especificación, diseño e implementación*, México, Alfaomega, 2002, pp. 462.

GOODRICH, Michael T. y TAMASSIA, Roberto, *Estructuras de Datos en Java*, México, Compañía Editorial Continental Grupo Patria Cultural, 2002, pp. 641.

JOYANES Aguilar, Luis, *et al, Estructuras de datos en C*, España, Mc Graw Hill, 2005, 435 pp.

KNUTH, Donald E., *El arte de programar ordenadores, Volumen I Algoritmos Fundamentales*, España, Reverté, 2002, pp. 672.

El arte de programar ordenadores, Volumen III Clasificación y Búsqueda, España, Reverté, 1987, pp. 776.

KRUSE, Robert L.**,** *Estructuras de Datos y Diseño de Programas*, México, Prentice Hall Hispanoamericana, 1988, pp. 488.

LANGSAM, Yedidyah, *et al, Estructuras de Datos con **C** y **C++***, México, Prentice Hall Hispanoamericana, 1997, pp. 672.

ROSE, César E., *Archivos, Organización y Procedimientos*, México, Computec, 1993, pp. 227.

WIRTH, Niklaus, *Algorithms + Data Structures = Programs* , New Jersey USA, Prentice Hall, 1976, pp. 366.

WIRTH, Niklaus, *Algoritmos y Estructuras de Datos*,México, Prentice Hall, 1987, pp. 305.

BIBLIOGRAFÍA COMPLEMENTARIA

BRAVO DE CAIRO, Carmen y STEINBERG Nosnik, Israel, *Tecninotas Cómo hacer eficientes los archivos en la B-6700, B-6800 Y B-7800,* México, Centro de Servicios de Cómputo CSC UNAM, Febrero de 1981, 11 pp.

DALE, NELL y Lilly Susan C., *Pascal y Estructuras de datos*, España, Mc-Grawl Hill, 1985, pp 491.

DEITEL, H.M. Y DEITEL, P.J., *Cómo programar en C/C++*, México, Prentice Hall, 1995, pp. 927.

REYES GONZÁLEZ, Armando, *Tecninotas Atributos de archivos,* México, Centro de Servicios de Cómputo CSC UNAM, Febrero de 1982, pp. 11.

ULLMAN JEFFREY D. y WIDOM, Jenifer, *Introducción a los sistemas de bases de datos,* México, Pearson, 1999, pp. 469.

VILLERS, Abelardo, *Tecninotas Introducción a bases de datos,* México, Centro de Servicios de Cómputo CSC UNAM, Septiembre de 1981, pp. 11 pp.

SITIOS DE INTERÉS

http://www.ujat.mx/oferta_educativa/dacbas/programas_sinteticos/matematicas/sustantiva_profesional/F0057_programacion2.pdf

http://www.ujat.mx/oferta_educativa/dacbas/programas_sinteticos/matematicas/sustantiva_profesional/F0057_programacion2.pdf

http://www.conclase.net/c/edd/index.php?cap=007

http://www.monografias.com/trabajos7/arch/arch.shtml

Soluciones a algunos ejercicios de las actividades complementarias al aprendizaje.

Actividad 2.6.

```
inicio
        indice=0
        i = √n
        p = 0
        k =1-i
        j = 1
        hacer mientras ( j ≤ i )
                p = p + 1
                k = k + i
                m = k
                c = k +1
                hacer mientras ( c≤ ( k + i – 1 ) )
                        si datos [c] < datos [m] entonces
                                m = c
                        fin
                        c = c + 1
                fin
                j = j +1
                ganadores [p][1] = datos [m]
                ganadores [p][2] = m
        fin
        j = 1
        hacer mientras ( j ≤ n )
                m = 1
                c= m +1
                hacer mientras ( c≤ i ) // se seleccionan el ganador de los
ganad.
                        si ganadores [c][1] < ganadores [m][1]
```

```
                        m = c
                fin
                c = c + 1
        fin
        p = m
        indice = indice + 1
        menor [indice] = ganadores [m][1]   // es el ganador de todos
        datos [ ganadores [m][2] ] = infinito
        k = i * ( m – 1 ) + 1
        m = k
        c= m +1
        hacer mientras ( c ≤  ( k + i – 1 )  )
                si datos [c]  < datos [m]
                        m = c
                fin
                c = c + 1
        fin
        ganadores [p][1] = datos [m]
        ganadores [p] [2] = m
        j = j + 1
    fin
fin
```

RESPUESTAS A LOS CUESTIONARIOS DE AUTOEVALUACIÓN. INFORMÁTICA III

Tema 1
1. *Archivo* es un conjunto de uno o varios registros semejantes, también se puede definir como una colección de datos relacionados entre sí, que pueden utilizarse de una misma forma.
2. Es el número de registros lógicos que caben en un registro físico.
3. Son los valores permitidos que pueden tomar esos datos y las operaciones que se puedan llevar a cabo sobre estos valores.
4. Entrada, salida y de entrada/salida
5. Reestructuración y reorganización.
6. Es la forma en la que los registros, se acomodan o almacenan en los dispositivos de almacenamiento o bien como se estructuran los datos dentro de un archivo.
7. Ver lo que hace el tipo de datos y no en el ¿Cómo lo hace?
8. El Absoluto, hacer referencia a direcciones físicas y el relativo, a direcciones relativas a la posición dentro del archivo.
9. El valor de la llave y la dirección relativa a los datos.
10. 1) La distribución actual de los valores de las llaves. 2) El número de registros con llaves diferentes que pueden ser almacenados en una dirección dada. 3) El número de valores de las llaves, que se estén utilizando, con respecto al tamaño de direcciones

Tema 2
1. Los métodos internos solo utilizan memoria principal, en tanto que los métodos externos, manejan memoria principal y secundaria.
2. Son: Selección, Intercambio, Inserción, Distribución y Mezcla.
3. El clasificar (ordenar) es acomodar estos elementos, para que queden de acuerdo a un orden preestablecido, el cual puede ser ascendente o descendentemente.
4. Siendo $K_1, K_2, K_3, . K_n$ (una lista de n elementos), entonces: Ascendentemente los valores de los datos quedan como: $K_1 \leq K_2 \leq K_3, \leq .. \leq K_N$
5. Descendentemente, los valores de los datos, de la lista K, quedan de la siguiente forma: $\mathbf{K}_1 \geq \mathbf{K}_2 \geq \mathbf{K}_3, \geq \ldots \geq \mathbf{K}_N$
6. De dos.
7. De árboles binarios o de Knuth.
8. No requiere ningún árbol, ya que solo maneja el arreglo o archivo original.
9. Se sugiere que el valor de h, sea de la mitad del número n, de elementos, ya sea la parte entera o redondeada, ya que si se toma por ejemplo un valor menor a esa parte entera, el algoritmo puede fallar.
10. Se recomienda que las h, se vayan decrementando a la mitad de su valor, pero para aumentar la

posibilidad del éxito del método, se sugiere que estos valores se decrementen de uno en uno.

Tema 3
1. Una colisión es la asignación de una dirección a un nodo, cuando ya ha sido asignada a otro por la misma función de Hash
2. Una función de Hash o de Mapeo, permite transformar a partir de la llave de un nodo a la dirección, en la cual se va a almacenar
3. Las formas de resolver colisiones son las direccionamiento Abierto y Encadenamiento.
4. Es un árbol en donde: - El dato que contiene cada uno de sus nodos, no se repite. a) El valor del dato que contiene la raíz de la rama izquierda (si es que existe), debe de tener un valor menor al de la raíz original. b) El dato de la rama derecha es mayor al de la raíz.
5. El recorrido Inorden
6. Comparación de llaves y transformación de llaves.
7. Secuencial (lineal) y Binaria.
8. El de transformación de llaves (Hash).
9. Cambiar de a una base numérica más pequeña, que la que esta utilizando originalmente la llave.
10. El de Búsqueda Binaria.

RESPUESTAS A LOS EXÁMENES DE AUTOEVALUACIÓN 1

INFORMÁTICA III

Tema 1	Tema 2	Tema 3
1. c	1. c	1. g
2. i	2. i	2. j
3. a	3. e	3. a
4. e	4. a	4. i
5. d	5. j	5. h
6. j	6. b	6. b
7. f	7. d	7. c
8. b	8. f	8. e
9. g	9. g	9. d
10. h	10. h	10. f
Segundo apartado		
1. a		
2. d		
3. b		
4. c		
5. a		

ANEXO A

TEMAS COMPLEMENTARIOS DE CLASIFICACIÓN

MEZCLA

Mezcla (Intercalación)

Estos métodos mezclan o juntan dos o más grupos de elementos, pudiendo ser estos grupos archivos (secuenciales generalmente), arreglos o simplemente listas, las cuales deben de estar previamente clasificadas bajo el mismo criterio ascendente ó descendentemente, no se puede dar el caso de mezclar una lista que se encuentre en forma ascendente con otra que esté en forma descendente, y mucho menos si están desordenadas.

Si las listas a mezclar están clasificadas en forma ascendente, la lista resultante también estará en orden ascendente y si las listas a intercalarse están en forma descendente, la lista resultante también estará en orden descendente.

Si se mezclan una lista A que tiene 10 llaves con otra lista B que tiene 15 elementos, la lista resultante C tendrá entonces 10 + 15 llaves.

Para realizar la mezcla, se lee el primer elemento de la lista A y el primeo de B y si están ordenadas en forma ascendente, el menor en valor es grabado en la nueva lista C y de donde se tomo el elemento que se grabó, se lee otro elemento y se siguen realizando las comparaciones y grabaciones a C de los elementos que van siendo más pequeños, hasta que se agote una de las listas y finalmente los elementos de la otra lista son pasados a la lista C y una vez agotadas todas las listas, termina el método.

Una variante que es la que se presentará a continuación es el realizar una mezcla con dos archivos secuenciales, donde se utilizará el tipo de movimiento que puede ser **a**lta, **b**aja o **c**ambio. El algoritmo que se presentará, consistirá en que se tiene tres archivos secuenciales, uno será el maestro1 que se

mezclará con el de movimientos, el cual indicara que tipo de movimiento afectará al archivo maestro1, pero como se tratan de archivos secuenciales, no es posible realizar la operación de rewrite por lo que se generará el nuevo archivo maestro2 que será igual al maestro1 afectado por los movimientos del maestro2.

El código es el que se presenta a continuación:

```
void mezcla(void)
{
   int bandera=1;

fscanf(maestro1,"%d%s%s",&reg_mto1.num_cta,&reg_mto1.nombre,&reg_mto1.
salario);
   if(feof(maestro1))
   {
      reg_mto1.num_cta=32767;
   }
fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
   if(feof(movimientos))
   {
      reg_mov.num_cta=32767;
   }

   while(bandera)
   {
      if(reg_mov.num_cta==reg_mto1.num_cta)
      /* Si se cumple esta condicion es cuando existe un registro en
el archivo
        maestro1 con el mismo numero de cuenta que en el archivo de
        movimientos.
      */
      {
       if (reg_mov.num_cta==32767) //Se trata del fin de los registros
                                   //de los dos archivos
           bandera=0;
       else
```

```
         if(reg_mov.tip_mov=='A')
        /* No se dara de ALTA y se leera unicamente otro registro del
archivo de
           movimientos  y no se grabara nada en el archivo maestro2
(el nuevo)
        */
        {
           fprintf(errores,"****No se puede dar de alta el cliente ya
existe****");
           fprintf(errores,"%d%s%s%c\n\n",reg_mov.num_cta,
reg_mov.nombre,reg_mov.salario,reg_mov.tip_mov);
fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
           if(feof(movimientos))
           {
              reg_mov.num_cta=32767;
           }
        }
        else if(reg_mov.tip_mov=='B')
        /* Este caso es cuando se dara una Baja de un registro que ya
existe y
           se procedera a no grabar nada en el archivo maestro2
(nuevo) y se
           se leera un registro del archivo maestro1 y otro registro
del archivo
           de movimientos.
        */
        {
fscanf(maestro1,"%d%s%s",&reg_mto1.num_cta,&reg_mto1.nombre,&reg_mto1.
salario);
           if(feof(maestro1))
           {
              reg_mto1.num_cta=32767;
           }

fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
           if(feof(movimientos))
           {
```

```
            reg_mov.num_cta=32767;
         }
      }
      else if(reg_mov.tip_mov=='C')
      /* Se trata de un caso donde existe un registro en el archivo
Maestro1 y
         se dara un cambio con el registro del archivo de
movimientos, por lo
         que se grabara el registro de movimientos al archivo
maestro2 (el nuevo)
         y se leera un registro del archivo de movimientos y otro
del archivo
         maestro1.
      */
      {

fprintf(maestro2,"%d%s%s\n",reg_mov.num_cta,reg_mov.nombre,reg_mov.sal
ario);

fscanf(maestro1,"%d%s%s",&reg_mto1.num_cta,&reg_mto1.nombre,&reg_mto1.
salario);
         if(feof(maestro1))
         {
            reg_mto1.num_cta=32767;
         }


fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
         if(feof(movimientos))
         {
            reg_mov.num_cta=32767;
         }
      }
   }
   else if(reg_mov.num_cta < reg_mto1.num_cta)
   /* Se trata de un registro del archivo de movimientos, que no
esta en
      en el archivo maestro1.
```

```
      */
      {
        if(reg_mov.tip_mov=='A')
        /* Es una ALTA que se realizara, escribiendo el registro de
movimientos
           al archivo maestro2 (nuevo)y despues se leera un registro
del archivo
           de movimientos.
        */
        {

fprintf(maestro2,"%d%s%s\n",reg_mov.num_cta,reg_mov.nombre,reg_mov.sal
ario);

fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
           if(feof(movimientos))
           {
              reg_mov.num_cta=32767;
           }
         }
         else if (reg_mov.tip_mov=='B')
         /* Es una baja que no se puede dar, ya que no existe en el
archivo
            maestro1, entonces se mandara el registro de movimientos
al archivo
            de errores y se leera otro registro del archivo de
movimientos
         */

         {
fprintf(errores,"%d%s%s%c",reg_mov.num_cta,reg_mov.nombre,reg_mov.sala
rio,reg_mov.tip_mov);
fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
           if(feof(movimientos))
           {
              reg_mov.num_cta=32767;
           }
```

```
         }
         else if(reg_mov.tip_mov=='C')
         /* Se trata de un CAMBIO que no se puede realizar, ya que no
existe
           en el archivo maestro1, se mandrara el registro de
movimientos al
           de errores y se leera otro del archivo de movimientos
(igual que
           en el caso anterior)
         */
         {
fprintf(errores,"%d%s%s%c",reg_mov.num_cta,reg_mov.nombre,reg_mov.sala
rio,reg_mov.tip_mov);

fscanf(movimientos,"%d%s%s%c",&reg_mov.num_cta,&reg_mov.nombre,&reg_mo
v.salario,&reg_mov.tip_mov);
           if(feof(movimientos))
           {
              reg_mov.num_cta=32767;
           }
        }
      }
      else if(reg_mov.num_cta > reg_mto1.num_cta)
      /* Se trata de un registro del archivo maestro, que no esta en
el archivo
        movimientos. O sea que ese registro del archivo maestro1 no
tiene
        ninguna afectacion y se grabara el registro del maestro1 al
maestro2
        (nuevo) y se leera otro registro del archivo maestro1.
      */
      {
fprintf(maestro2,"%d%s%s\n",reg_mto1.num_cta,reg_mto1.nombre,reg_mto1.
salario);
fscanf(maestro1,"%d%s%s",&reg_mto1.num_cta,&reg_mto1.nombre,&reg_mto1.
salario);
        if(feof(maestro1))
        {
           reg_mto1.num_cta=32767;
```

```
        }
      }
   }
} // fin de la funcion mezcla
```