# TEMA I: ARCHIVOS

## CONCEPTOS IMPORTANTES

### *Archivo*

***Definición 1***: es un conjunto de registros homogéneos que contienen datos heterogéneos.
***Definición 2***: es un conjunto de registros homogéneos bajo una determinada organización (o bajo una estructura en particular.)
***Definición 3***: es un espacio de almacenamiento direccionable, nominable (con nombre), que contiene datos o información bajo una determinada organización o estructura.
***Definición 4***: es un conjunto de información localizada y almacenada como una unidad (posee nombre y dirección de ubicación).
***Definición 5***: es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros o artículos (relacionados entre sí) que son de igual tipo y que constan, a su vez, de diferentes entidades de nivel más bajo denominadas campos.

### *Registro*

***Definición 1****:* es cada uno de los componentes del archivo que posee una cierta estructura idéntica para todos ellos con objeto de almacenar la información referente al tema general del archivo.
***Definición 2***: es un conjunto de campos relacionados que contienen elementos o datos elementales. Todos los registros tienen la misma estructura.
***Definición 3: Registro (Record)*** es una colección de campos relacionados que pueden tratarse como una única unidad por un programa de aplicación. Por ejemplo: un registro de empleados va a contener campos como nombre, número de empleado, DNI, ciudad, etc.

Dependiendo del diseño, los registros pueden ser de longitud fija o de longitud variable.

- ***Registros de longitud fija:*** siempre tienen el mismo tamaño, esto ahorra tiempo de procesamiento, son más fáciles de manejar y cubren las necesidades de la mayoría de las aplicaciones. Sin embargo, no siempre hacen uso eficiente del espacio asignado, no todos los valores asignados a los campos ocupan el espacio reservado.
- ***Registros de longitud variable:*** el tamaño del registro puede variar debido a que los datos varían en longitud o debido a que el número de datos cambie de un caso a otro.

### *Campo*

***Definición 1***: es una mínima parte constitutiva de los registros.
***Definición 2***: es la unidad mínima y elemental de información accesible en un registro.
***Definición 3:*** es la unidad mínima de información de un registro, o también se dice que es un ítem o elemento de datos elementales, tal como un nombre, número de empleado, ciudad, DNI, etc.
***Definición 4:*** (Field) es el elemento de datos básico. Un campo individual contiene un valor único. Está caracterizado por su longitud y por el tipo de datos. Dependiendo del diseño del archivo, los campos pueden ser de tamaño fijo o variable. Los datos contenidos en un campo se pueden dividir en subcampos; por ejemplo, el campo *fecha* se divide en los subcampos *día*, *mes*, y *año.*

### *Clave*

***Definición 1:*** una clave (key) o *indicativo* es un campo de datos que identifica al registro y lo diferencia de otros registros. Esta clave se utiliza para facilitar la extracción de información de un registro. Se espera identificar en forma unívoca cada registro almacenado en el archivo. Claves típicas son nombres o números de identificación. Condición necesaria y suficiente de la clave: unicidad y minimalidad, una clave por registro es la minimalidad y el dato de esa clave debe ser único, por ejemplo, DNI.

## CLASIFICACION DE LOS ARCHIVOS DE ACUERDO AL CICLO DE VIDA

Los archivos se clasifican en persistentes y temporales.

- **PERSISTENTES:** se almacenan generalmente en memoria secundaria (discos rígidos, CD, etc.) y permanecen en ella desde su creación hasta su eliminación explícita.
- **TEMPORALES:** pueden residir en memoria secundaria o en memoria principal, y tienen un tiempo de vida determinado de acuerdo a la ejecución de uno o más procesos.

## CLASIFICACION DE LOS ARCHIVOS DE ACUERDO A LA INFORMACIÓN CONTENIDA

De acuerdo a la información contenida en los archivos, éstos se clasifican en:

- **Archivos de programa:** almacenan instrucciones para manipular los datos.
- **Archivos de datos:** almacenan datos en sentido estricto (en forma de registros).
  - ***Archivo Maestro:*** conjunto de registros acerca de un aspecto importante de las actividades de una organización. Pueden contener datos que describen el estado actual de eventos específicos o indicadores de la empresa. Son útiles sólo mientras se mantengan exactos y actualizados. Antes de que los archivos puedan utilizarse, deben ser mantenidos para reflejar, incluso, los eventos más recientes que afecten los datos en ellos. Esto se logra mediante el uso de Archivos de Transacciones (eventos que afectan a la organización y sobre los cuales se calculan datos).
  - ***Archivo de Transacciones***: archivo temporal con dos propósitos: acumular datos acerca de los eventos al momento que ocurran y actualizar los archivos maestros para reflejar los resultados de las transacciones actuales. Los archivos maestros son permanentes y duran mientras exista el sistema. Sin embargo, los contenidos de los archivos cambian como resultado del procesamiento y actualización. Por otro lado, los archivos de transacciones son temporales, cuando no son necesarios se borran o se destruyen.
  - ***Archivo de Reportes:*** archivo temporal que se utiliza cuando en tiempo de impresión no está disponible para todos los reportes producidos, situación que surge con frecuencia en el procesamiento superpuesto. La computadora escribe el reporte a un archivo en disco, donde permanece hasta que pueda imprimirse. Este proceso se conoce como impresión por cola; es decir, la salida que no puede imprimirse cuando se produce, forma una cola en un archivo de reportes, en espera de respuesta a la solicitud de impresión. Los archivos de reportes se pueden utilizar con muchos otros dispositivos se salida.
  - ***Archivo de Respaldo:*** copia de un archivo maestro, o de transacciones, hecho para garantizar que se dispone de un duplicado si algo le ocurre al original.

## ORGANIZACION DE LOS ARCHIVOS

Se entiende por organización de los archivos a la forma en que se estructuran dentro del dispositivo de almacenamiento.

***Métodos de organización de archivos:***

El número de alternativas posibles en la organización de archivos es casi ilimitado. A continuación, se describen cuatro alternativas de diseño básico de archivo.

- ***Archivo secuencial***
- ***Archivo indexado***
- ***Archivo secuencial indexado***
- ***Archivos directo.***

### • ARCHIVO SECUENCIAL

**Descripción:** surgen de una organización llamada Apilo. Es la forma más simple de almacenar y recuperar registros en un archivo. Un fichero secuencial está organizado físicamente de tal forma que cada registro es adyacente al siguiente registro del mismo fichero; es decir, que los registros se almacenan uno tras otro ocupando espacios de memoria contiguos. No existen posiciones sin uso. No

existe la posibilidad de acceder directamente a los registros y se accede a ellos en el orden en que fueron escritos en el fichero, del primero al último y de uno en uno.
Para localizar un registro es necesario examinar cada registro hasta encontrar el deseado.

| | Nombre | Edad | Estatura | IQ |
|---|---|---|---|---|
| **1** | Antwerp | 55 | 5′8″ | 95 |
| **2** | Berringer | 39 | 5′6″ | 75 |
| **3** | Bogley | 36 | 5′7″ | 70 |
| **4** | Breslow | 25 | 5′6″ | 49 |
| **5** | Calhoun | 27 | 5′11″ | 80 |
| **6** | Finnerty | 42 | 5′9″ | 178 |
| **7** | Garson | 61 | 5′6″ | 169 |
| **8** | Pop | 36 | 5′7″ | 83 |
| **9** | Purdy | 31 | 5′6″ | 95 |
| **10** | Young | 59 | 5′5″ | 145 |
| **11** | Kroner | 26 | 5′8″ | 47 |
| **12** | Moskowistz | 27 | 5′2″ | 75 |

Archivo Secuencial

Una operación de lectura o escritura lee o escribe el registro y avanza al apuntador al siguiente registro.

- **Altas:** el nuevo registro se agrega a continuación del último dado de alta (altas al final).
- **Bajas:** no hay forma física de dar de baja un registro intermedio y realizar el corrimiento; en estos casos se debe hacer una baja lógica. La única forma de realizar la baja física es generar un archivo nuevo sin el registro marcado; el archivo original se borra.
  Una baja lógica es marcar de alguna manera un registro dentro del archivo secuencial para no utilizarlo. Un ejemplo podría ser dar de baja un registro si el campo DNI es igual a 0.

| D.N.I. | APELLIDO | NOMBRE | CIUDAD | … |
|---|---|---|---|---|
| 23114589 | Benítez | Claudia | Oro Verde | … |
| 0 | Mitivié | Carlos | Cerrito | … |
| 27415777 | Boló | Hernán | Viale | … |
| 23066774 | Bertolo | Pablo | Paraná | … |
| … | … | … | … | … |

Las bajas lógicas las define el programador.

- **Modificaciones:** los sistemas más viejos no permitían realizar modificaciones. Actualmente, se permite reescribir un registro en un archivo secuencial.
- **Acceso:** los registros se acceden por su orden de aparición en el archivo. Para localizar un registro específico se utiliza una clave de búsqueda que es comparada con cada registro del archivo, comenzando desde el principio. Este proceso de lectura y comparación se repite hasta encontrar un valor coincidente, de lo contrario continua hasta alcanzar el final del archivo (EOF –End Of File).
  - Costo en fracaso = $n + 1$ ($n$ es la cantidad de registros).
  - Costo promedio = $\frac{n+1}{2}$

**Nota:** si el arreglo está ordenado según frecuencia de uso y la búsqueda es secuencial, entonces tenemos un costo en fracaso = $n$ y un costo en éxito = $\frac{n}{\ln n}$

- **Ventajas y desventajas:** es el más óptimo en cuanto al aprovechamiento de memoria, ya que se ocupa espacio a medida que se necesita, no requiere espacio previo inicial.

Si es necesario acceder a cada registro en el archivo para una aplicación particular, un archivo secuencial es un buen método de organización. Si, en promedio, alrededor de la mitad de los registros en el archivo se van a utilizar, la organización secuencial sigue siendo aceptable, ya que al menos uno de dos registros recuperados será utilizado y el tiempo necesario para examinar un registro adyacente es breve.

Por otro lado, si el requerimiento es hallar un registro particular en un archivo muy grande, la organización secuencial del archivo se convierte en una desventaja. El programa debe comenzar al principio del archivo y leer cada registro hasta encontrar el correcto, una tarea que consume mucho tiempo, a menos que el registro deseado sea uno de los primeros en el archivo.

En general, si se necesitan menos del 10% de los registros en un archivo durante una ejecución común de procesamiento, el archivo no debe establecerse como un archivo secuencial. Por otro lado, si se desea acceder más del 40% de los registros, el analista debe elegir la organización secuencial. Entre el 10% y el 40%, la decisión depende del tamaño del archivo, la frecuencia de uso, o si será actualizado con frecuencia o utilizado sólo para recuperación.

Para las aplicaciones interactivas que incluyen peticiones o actualizaciones de registros individuales, los archivos secuenciales ofrecen un rendimiento pobre.

Otra desventaja es el costoso mantenimiento de este tipo de organización.

- **Utilización de archivos secuenciales**: los archivos secuenciales son el tipo de archivo utilizado con mayor frecuencia en el procesamiento comercial de datos orientado al manejo por lotes.

  Cuando los datos se procesan cíclicamente, como en el caso de las aplicaciones mensuales de facturación o nómina, la efectividad de este enfoque es difícil de lograr mediante otros métodos. Sin embargo, los datos conservados en archivos secuenciales son difíciles de combinar con otros datos para proporcionar información con algún propósito específico, y es necesario el manejo cronológico del acceso al archivo, si la información solicitada debe ser actual.

  La organización secuencial es conveniente para el procesamiento exhaustivo de todos los registros de un fichero (por ejemplo, cálculo de nóminas, listados completos de empleados, actualización de parámetros, recuento de registros, cálculo de medias, varianzas, etc.).

  Además, el uso de los archivos secuenciales es idóneo para aplicaciones fuera de línea (*off line*).

  Ejemplo de organización secuencial: casette.

## ORGANIZACIÓN INDEXADA

**CARACTERISTICAS GENERALES:** Representa una especie de balance entre la organización de archivos secuenciales y archivos directos relativos (directos), ya que el método usa un examen secuencial del índice, seguido del acceso directo a la dirección física apropiada en el área de datos.

El principio fundamental de las organizaciones y métodos de acceso indexados es el de asociar a la clave de un registro su dirección relativa dentro del fichero mediante una 'tabla de contenido' del fichero, llamado índice. El índice contiene las claves de todos los registros asociadas a la dirección relativa de cada uno de ellos. La tabla de índices puede almacenarse en el mismo fichero o en un fichero separado.

Un archivo con organización indexada consta de un área de índice y un área de datos.

El ***área de datos*** es el área de almacenamiento que contiene a los registros y está direccionada por el área de índice.

El ***área de índice*** incluye una clave de registro y la dirección de almacenamiento asociada; dicha dirección es el lugar físico donde se encuentran los datos o información del registro en particular.

El área de índice puede considerarse como una tabla pero en realidad generalmente se implementa en un Árbol B, en el cual sus ramas son información de cómo se van generando los índices y los nodos terminales tienen las direcciones físicas del registro asociado a la clave. Esto se debe a que la estructura de árbol es más rápida que una tabla para la administración de claves. El área de índices es de constante actualización ya que debe respetar un orden en cuanto a la ubicación de las claves (generalmente ascendente). Un archivo puede tener más de un índice y éste puede tener uno o varios campos claves.

El fichero indexado puede contener las claves de todos los registros o solamente un subconjunto de ellas. Se llama densidad del índice al cociente resultante de dividir el número de claves contenidas en el índice entre

el número de registros del fichero. Si el índice contiene las claves de todos los registros del fichero, decimos que el índice es denso, y si la densidad del índice es menor que la unidad, decimos que el índice es no denso.
Si el índice es no denso, los registros se almacenan ordenadamente en bloques conteniendo el índice, la clave mayor contenida en el bloque y la dirección del bloque para cada bloque del fichero.
Los índices ocupan espacio adicional pero proporcionan un método rápido de localización de registros.
Es más rápido que el archivo secuencial pero más lento que el direccionamiento directo.
Flexibilidad para procesar el archivo en cualquier forma dependiendo del requerimiento en el momento.
Es conveniente utilizar un archivo indexado cuando se producen muchas variaciones de volumen, ya que no es necesario dimensionarlo previo a su creación.
Cuando se crea un archivo como indexado, los registros deben estar ordenados con respecto a su clave y el administrador de archivos crea tanto un área de datos como el área de índices.
Para hallar un registro cuando se desconoce la dirección de almacenamiento, es necesario examinar todos los registros; sin embargo, la búsqueda será más fácil si se usa un índice, ya que toma menos tiempo buscar en un índice que en un archivo completo de datos.
El procedimiento a seguir para localizar un registro específico es:

1. Leer el índice.
2. Ubicar la clave.
3. Tomar la dirección
4. Buscar el dato en el área de datos.

## • ARCHIVO INDEXADO PURO

➲ **Descripción:** consta siempre de dos áreas: un área de índices y un área de datos. El área de datos puede estar estructurada en espacios contiguos o bien cada registro estar distribuido de forma individual. Lógicamente es indistinto, ya que de todas formas se conserva un nexo entre los registros cuando están separados. El área de índice almacena una clave y una dirección asociada a esta clave por cada registro en el área de datos.
Una clave es un conjunto de uno o más campos pertenecientes al registro del archivo que identifica unívocamente al registro (claves únicas). Para que un conjunto de una o más claves pueda ser clave, debe cumplir con la unicidad y la minimalidad.

- **Unicidad:** que una clave pertenezca a un registro, que no existan duplicaciones y que no tenga valores nulos. Dicho de otra manera, no hay dos claves con el mismo valor.
- **Minimalidad:** si la clave es compuesta (más de un campo), cualquier campo que se quite de la clave hace perder el criterio de unicidad.

El área de claves es un área de búsqueda organizada generalmente con un Árbol B, por la clave propiamente dicha.

**Área de Índice**

| Clave de Registro | Dirección |
|---|---|
| AB | 1021 |
| AC | 1021 |
| AD | 1022 |
| BC | 1018 |

**Área de Datos**

| | | | |
|---|---|---|---|
| 1021 | 24613787 | Alarcón Torres | Andrea |
| 1021 | 24592208 | Befani | Diego |
| 1022 | 84507114 | Alarcón Torres | José |
| 1018 | 24300116 | Schunck | Mariana |

➲ **Altas:** físicamente, el administrador de archivos ubica la nueva clave en el área de índice, consulta y reserva una dirección en el área de datos, que asocia en el área de índice, luego graba la información del registro en el área de datos reservada, y por último establece el encadenamiento del nuevo registro con los existentes.

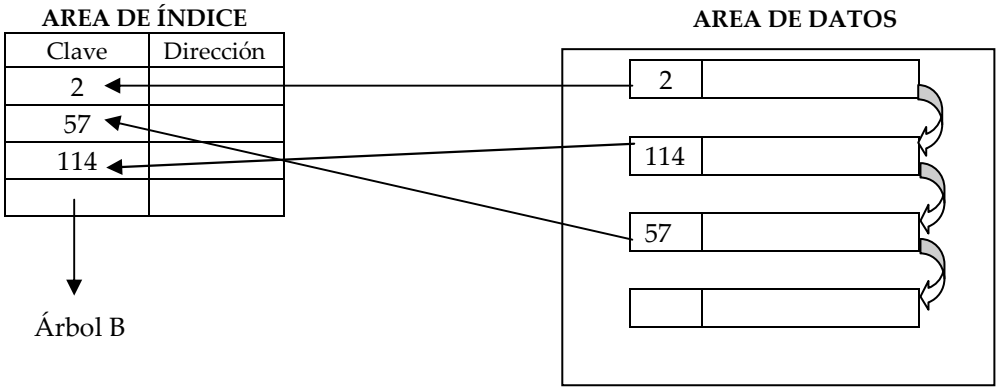
➲ **Bajas:** permite las bajas físicas (área de índice) y lógicas (área de datos). Una baja física puede realizarse de dos formas:

- se busca la clave y se obtiene la dirección de los datos. En el área de índice se realiza una baja física, y en el área de datos se marca el registro. No se pierde el registro pero tampoco se puede utilizar, dado que se pierde la dirección para localizarlo. Para evitar desperdiciar espacio de memoria y mantener actualizado el archivo, es necesario depurarlo. Haciendo uso del encadenamiento, se verá qué registros están dados de baja lógicamente y luego se convertirá en memoria disponible ese sector y se deben modificar los enlaces correspondientes para la reorganización del archivo (primero se da de baja en el AD, luego en el AI).
- Primero se ubica el registro en el área de datos, se libera el lugar que ocupaba y recién luego se elimina físicamente la clave del área de índice.

Cuando el archivo no está en uso, el área de índice posee dos lugares de trabajo, en el disco (medio magnético, lugar físico) y en la memoria principal (área de trabajo por excelencia).

- **Modificaciones:** se puede regrabar directamente sobre el registro. Se ubica la clave en el área de índice, se localiza el registro en el área de datos, el que se carga en memoria para ser modificado y posteriormente grabado. Si se cambia la clave, se debe reorganizar el área de índice.
- **Acceso:** la única forma de acceder a un registro de un archivo indexado es a través de una clave. Dada una clave, se la busca en el área de índice; si se la encuentra, se busca la información en el área de datos.
- **Utilización de archivos indexados**: se utilizan en sistemas *on line*; por ejemplo, dentro de la facultad, en el sistema de alumnado. Se usan principalmente en áreas en las que la oportunidad temporal de la información sea muy crítica. Se encuentran ejemplos de esto en los sistemas de reservación de las aerolíneas, en los trabajos bancarios, en los sistemas militares de datos y en otras aplicaciones de tipo inventario. En estos casos, es raro que los datos se procesen serialmente, excepto en las revisiones de existencias ocasionales, tal vez exclusivamente anuales.

  Otras instancias en las que resultan convenientes los archivos indexados se presentan cuando los datos son sumamente variables y dinámicos.




## RECONSTRUCCION DE UN ARCHIVO INDEXADO

Es factible la pérdida de información por cortes de energía si se está trabajando en Memoria Principal, o por rupturas de pistas si la información está alojada en disco.

Si se pierde el área de índice completa, no existe forma de acceder, es por ello que todo administrador de archivos prevé una regeneración en el área de índices.

Para reconstruir el archivo se utiliza una herramienta llamada regenerador que va directamente al área de datos.

Si es contigua y la clave es parte del registro, arma el área de índice con la clave y la dirección; si en el registro no está la clave nunca podría generarse el área de índice. Si el espacio no es contiguo, es fundamental tener la dirección del primer registro hasta llegar al último; es importante guardar la clave en el registro de datos para poder reparar el índice; una vez realizado esto se habrá generado el área de índice.

Es fundamental tener la dirección del primer registro, por eso todo administrador de archivos lo hace (en DOS se guarda en la FAT, en FOX → REINDEX, en COBOL → RECOVERY).

♦ Nota: si se pierde la dirección del primer registro, se podría recuperar manualmente trabajando con las direcciones reales (DEBUG).

## • ARCHIVO SECUENCIAL INDEXADO (ISAM: Indexed Sequential Access Method)

➲ **Descripción:** Los ficheros secuenciales indexados se organizan por bloques de registros donde los índices apuntan a cada bloque. Un fichero ISAM consta de tres zonas lógicas:

- **Área primaria de datos**, donde se almacenan los registros.
- **Área de desbordamiento**, también llamada de desborde, de *overflow* o de derrame, donde se almacenan los registros cuando se llena el área primaria de datos.
- **Área de índice,** donde se almacenan los índices.

El área primaria está organizada por bloques, es decir, almacena bloques individuales de registros (en orden secuencial). Dentro de cada bloque hay una cantidad fija de registros, ordenados por clave. Esa cantidad (n) de registros la determina el administrador de archivos. Los registros contenidos en los bloques están dispuestos en forma consecutiva. El acceso a cada bloque es directo, y en el bloque se busca el registro (clave) secuencialmente.
El área de índice sólo almacena las direcciones físicas correspondientes al comienzo de cada bloque y su clave asociada. Es necesario sólo un índice para cada bloque.
El área de saturación se utiliza para almacenar aquellos registros que en las actualizaciones no han tenido cabida en el área primaria. Posee una organización secuencial.
El área de desborde aparece cuando los archivos han sido dimensionados; sirve para cubrir todas las posibilidades de fallas en el dimensionamiento.
Generalmente, la dimensión del área de overflow es un 15% o 20% del área de datos.
Si el área de desborde crece mucho, quiere decir que el dimensionamiento elegido no fue el correcto. Si permanece estable, implica que el dimensionamiento es adecuado.

**AREA DE INDICE**

| Clave del registro | Dirección de comienzo del bloque |
|---|---|
| 1115 | 1345 |
| 1315 | 1349 |
| 1429 | 1346 |
| 1725 | 1350 |

**AREA DE DATOS**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1345** | 1010 | 1011 | 1013 | 1014 | 1017 | 1019 | 1110 | 1113 | 1115 | | | | | |
| **1346** | 1316 | 1317* | 1321 | 1323 | 1324 | 1410 | 1414 | 1415 | 1417 | 1418 | 1419 | 1427 | 1428 | 1429 |
| **1349** | 1117 | 1120 | 1121 | 1210 | 1211 | 1212 | 1215 | 1217 | 1218 | 1221 | 1310 | 1313 | 1315 | |
| **1350** | 1510 | 1521 | 1522 | 1617 | 1619 | 1620 | 1721 | 1724 | 1725 | | | | | |
| **O- F-** | *1318 | | | | | | | | | | | | | |

➲ **Altas:** si no existe el archivo, se crea un bloque (definiendo su tamaño y su porcentaje de ocupación) y se graban los datos con la clave en el primer lugar del bloque; por último, se actualiza el área de índice

con la clave y la dirección del bloque. Si existe el archivo, se debe buscar en qué bloque cabría la clave a dar de alta considerando que si el por ciento de ocupación es mayor al previsto se abre un nuevo bloque y se inserta el registro, esto se realiza así para asegurar que siempre exista lugar. Luego se debe indicar en el índice que esa clave está en el nuevo bloque, lo que implica incrementar el área de índice cuando se dan de alta nuevos bloques.

Al querer adicionar registros y no contar con el espacio suficiente para reacomodar los registros dentro del bloque y mantener el orden secuencial con respecto a su clave, se crea un área de saturación. De este modo, físicamente el archivo principal ya no es secuencial; sin embargo, desde el punto de vista del programador, el archivo es lógicamente secuencial. Los registros en las pistas están encadenados a los registros en las pistas de saturación, para que el archivo sea accesible como si estuviera en orden físico secuencial.

Dentro de cada bloque se reserva un GAP de protección (30/40%); una vez cubierto el resto del bloque, un alta de clave superior a la última clave genera la apertura de un nuevo bloque, quedando el GAP de protección para los nuevos registros que tengan el valor entre la mínima del bloque y la mínima del siguiente bloque.

Una vez que el bloque se llena, todos los nuevos registros correspondientes a ese bloque se graban en el área de desborde.

En síntesis, para dar de alta un nuevo registro, se siguen los siguientes pasos:

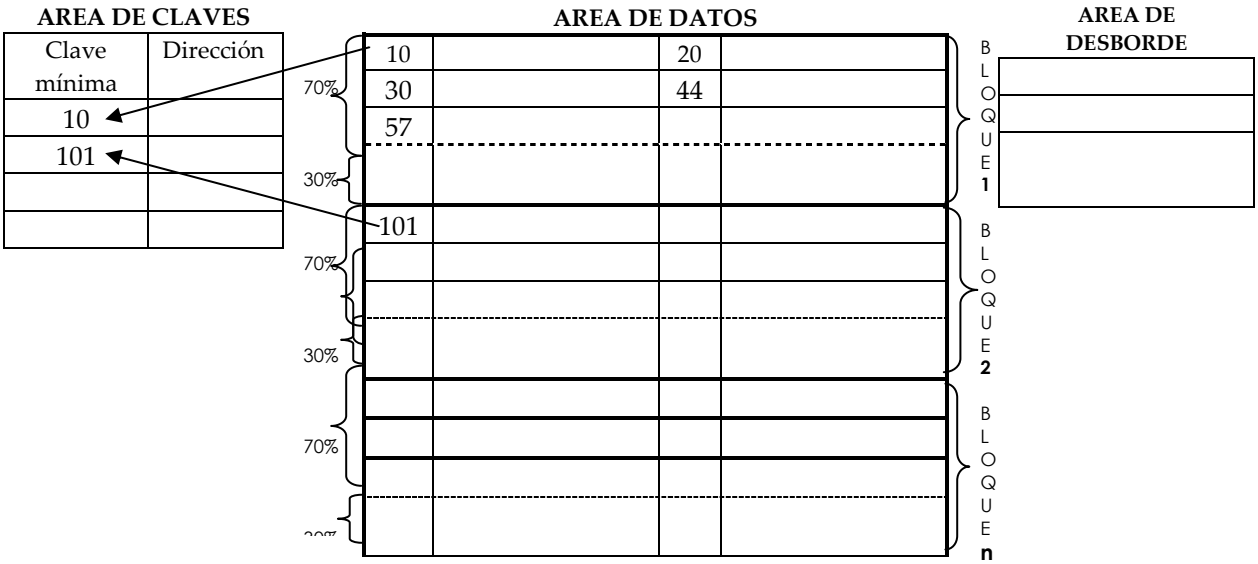
- Se recorre hasta encontrar una clave mayor, sabiendo así que el nuevo registro se ubicará en el bloque anterior.
- Luego de tener la dirección del bloque, se graba el registro al final del mismo.
- Si se tiene lugar en el bloque de datos, se da de alta.
- Si no hay lugar dentro del bloque, se lo graba en el área de desborde, actualizando los links.

- **Bajas:** se puede marcar lógicamente el registro dentro del bloque, o bien producir un corrimiento. En el caso que el registro borrado sea el mismo del índice (clave), deberemos modificar el área de índices.
  Se utilizan los bloques en memoria, se recorren en memoria.
  Si el registro es único, desaparece el bloque y su referencia en el área de índices, es decir, que sólo es posible dar de baja físicamente si es el registro del bloque.
- **Modificaciones:** se puede regrabar directamente sobre el registro. Para localizar un registro, en primer lugar el programa accesa al índice, empezando con el registro del principio, compara la clave en el índice con la clave de búsqueda, hasta ubicar la clave correspondiente al bloque en el que se encuentra el registro a modificar. El programa le dice al sistema que comience a leer registros en la dirección asociada a la clave y compara la clave de cada registro con la clave de búsqueda. Cuando éstas coinciden, se cargan los datos en memoria para ser modificados y posteriormente grabados.
  Si se cambia la clave, se debe reorganizar el área de índice
- **Acceso:** Para localizar un registro en el fichero, se accede a través de un índice al bloque que contiene el registro, y mediante una búsqueda secuencial se localiza el registro en el bloque.
- **Ventajas y desventajas:** Las ventajas del método vienen del hecho de que el fichero está ordenado, lo que facilita el acceso secuencial en orden creciente y los tiempos de acceso son buenos siempre que el fichero no esté desbordado.
  Este tipo de organización de ficheros es muy utilizada porque mantiene un buen equilibrio entre el tamaño de la tabla de índices y el tiempo de acceso a los registros.
  En cuanto a las desventajas, los inconvenientes son varios. Hay que mencionar que se asigna espacio de almacenamiento que puede no ser utilizado dentro de un bloque. Por su parte, la gestión de desbordamiento es compleja y degrada las prestaciones hasta hacer necesaria una reorganización periódica de los ficheros.
  El área de saturación puede crecer excesivamente, lo que supone una disminución en la rapidez en su tratamiento y una mayor dificultad en su procesamiento.
- **Utilización de archivos secuenciales indexados**: se utilizan en trabajos *on line* para grandes volúmenes de datos. Los archivos secuenciales indexados del tipo básico analizado anteriormente se encuentran en uso común en el moderno procesamiento comercial. Se utilizan en forma especial cuando existe la

necesidad de conservar archivos actualizados dentro de marcos de tiempo menores que los intervalos del procesamiento factibles con la reorganización cíclica de archivos secuenciales.

Este tipo de archivo es adecuado para el acceso en línea u orientado a terminales. Esto no es posible con los archivos de tipo secuencial.

También se utilizan comúnmente para manejar consultas, con la restricción de que la consulta deberá especificar el atributo llave. Son casos comunes de este empleo las consultas de facturación basadas en números de cuenta.




## • ARCHIVOS DIRECTOS

- **Descripción:** Medio u organización que permite procesar o acceder en forma rápida a los registros haciendo referencia directamente a su posición relativa en el soporte de almacenamiento, sin necesidad de pasar por la información anterior. Son el único tipo de archivo donde su acceso es directo.

  Se debe establecer la dimensión (longitud y cantidad de registros) previo a su utilización. Esto nos obliga a hacer una estimación sin saber la cantidad de registros que se van a utilizar. Para dimensionarlo, hay que tener en cuenta:
  - Cantidad de registros a almacenar
  - Longitud de cada registro.

  Cada registro tiene asociado un **Número Relativo de Registro** (numeración de los registros), que no es la dirección física que tiene realmente en el disco, es el Sistema Operativo quien se encarga de relacionar el Nº Relativo de Registro con la dirección física.

  Todo registro posee un campo de identificación (clave). Es conveniente al guardar la información que este campo esté en relación con el número relativo de registro, a fin de poder acceder por medio de él.

  No es necesario su orden. Como todo tipo de archivo dimensionable, posee un área de overflow o de desborde, que se define en el mismo momento de definir al área de datos.
- **Altas:** para dar de alta a un registro, es necesario posicionarse según el número relativo de registro y luego grabar la información, siempre haciendo referencia al número de registro. Como se accede directamente a los N registros, entonces es posible dejar espacios entre uno y otro. Si se accede por simulación de acceso secuencial, se da de alta como en archivos secuenciales.
- **Bajas:** las bajas son lógicas, pero es posible crear un archivo de menor dimensión e igual tipo, luego copiar los registros que no están marcados como dados de baja en éste, para finalmente borrar el archivo de origen; o se puede usar el campo marcado lógicamente para determinar cuando es posible sobrescribir los datos de un registro (que lógicamente está borrado).
- **Modificaciones:** Para realizar modificaciones se debe ubicar el registro, leerlo y una vez disponibles los datos en memoria hacer los cambios pertinentes para luego reposicionar el registro y grabarlo.

- **Acceso:** ante la petición de un registro determinado, el método calcula la dirección del bloque físico que lo contiene y accede a él directamente. El acceso es rápido y tiene costo 1.
  También es posible acceder al registro mediante el contenido de su campo clave si éste corresponde exactamente a un Nº Relativo de Registro (NRR). Sin embargo, ya que no es usual que las claves de registro cumplan este requerimiento, a menudo se utiliza el *Hashing* para proporcionar un medio de acceso directo. Con Hashing, se obtiene un NRR aplicando una función matemática de transformación de la clave que divide, extrae o dobla la clave del registro. Es necesario que el número relativo de registro esté en un rango determinado con relación a la dimensión del archivo (1 – N) y que los valores devueltos no se repitan, de lo contrario no se puede asegurar que se refiere a un registro en especial. Además, los valores obtenidos deben distribuirse de manera uniforme en todo el espacio asignado en vez de acumularse todos juntos; esto garantiza una recuperación más rápida de los registros y hace un mejor uso del espacio.
  Si no se puede trasformar una clave porque no se tiene una función para transformarla, se simula una lectura secuencial, que consiste en acceder al primer registro del archivo, leer los datos y determinar, mediante una comparación, si es el buscado; si no lo es, se incrementa en uno la posición y se compara nuevamente; así sucesivamente hasta llegar al registro deseado. Este procedimiento de búsqueda es muy lento (incluso más que el secuencial).
- **Ventajas y desventajas:** el método de acceso directo es rápido, ya que se evitan las operaciones intermedias de archivo. Si se puede asociar una clave de registro con un NNR, el archivo directo es 100% eficiente. Si no se puede hacer la coincidencia, el archivo prácticamente no sirve. Entre no servir y el ideal, existe el Hashing. Por ejemplo, teniendo 100 proveedores (con número de proveedores contiguos), se asocia Número de proveedor 1 con NNR 1, y así sucesivamente. En cambio, por ejemplo con DNI, no resulta eficiente.
- **Utilización de archivos directos**: se usan cuando la información no tiene gran variabilidad, tanto los números de registros como en el contenido de cada registro. En aplicaciones en que los tamaños de registro son pequeños y fijos, donde el acceso rápido es esencial y donde el acceso a los datos siempre se efectúa en forma simple, la organización de archivo directo es muy adecuada. En este caso, acceso simple significa emplear una sola llave para recuperar, y no acceso en serie. A modo de ejemplo, los archivos directos se utilizan con frecuencia en directorios, tablas de precios, programación de tiempos, listas de nombres, etc.
  Los archivos directos también desempeñan un importante papel como componentes de organizaciones más complejas de archivos.

## TRASPASO DE INFORMACION ENTRE DISTINTOS ARCHIVOS

### *Traspaso de la información de un archivo secuencial a uno directo*

1. Abrir el archivo secuencial.
2. Inicializar un contador de registros.
3. Leer un registro del archivo secuencial.
4. Actualizar contador de registros.
5. Repetir los pasos 3 y 4 hasta encontrar la marca de fin de archivo.
6. Cerrar el archivo secuencial.
7. Dimensionar el archivo directo sabiendo que: Dimensión = Contador de Registros * longitud de registros. Los registros del archivo directo deben poseer la misma estructura que los del archivo secuencial.
8. Abrir el archivo secuencial.
9. Recuperar un registro del archivo secuencial.
10. Grabar la información leída del archivo secuencial en el archivo directo, haciendo referencia al número relativo de registro, el cual se puede llegar a obtener a través de alguna transformación de la clave del archivo secuencial (Por ejemplo: Hashing).
11. Repetir los pasos 9 y 10 hasta que finalice la lectura de todos los registros del archivo secuencial (fin de archivo-EOF).
12. Cerrar los dos archivos.

### *Traspaso de la información de un archivo secuencial a uno indexado*

1. Abrir el archivo secuencial.
2. Crear el archivo indexado (área de datos y área de índice), con registros que posean la misma estructura que los del archivo secuencial. (Puede ser necesario agregar un campo para la clave).
3. Recuperar un registro del archivo secuencial.
4. Ubicar la nueva clave en el área de índice del archivo indexado manteniendo el ordenamiento ascendente o descendente de esta área (como clave se puede utilizar algún dato del registro del archivo secuencial, por ejemplo un número correlativo –la clave deber ser única–).
5. Reservar una dirección en el área de datos.
6. Grabar la información leída del archivo secuencial en la dirección reservada en el área de datos.
7. Asignar la dirección del área de datos reservada al índice recién creado en el área de índice.
8. Encadenar el nuevo registro a los demás registros del área de datos, si es que no están en espacios contiguos.
9. Repetir los pasos 3, 4, 5, 6, 7 y 8 hasta encontrar la marca de fin de archivo en el archivo secuencial.
10. Cerrar los dos archivos.

### *Traspaso de la información de un archivo directo a uno secuencial*

1. Abrir el archivo directo.
2. Haciendo uso de una función específica de los archivos directos, obtener la longitud del archivo. Si no se cuenta con dicha función, necesariamente se debe conocer la dimensión del archivo y longitud de cada registro para obtener la cantidad de registros por medio del cálculo: dimensión del archivo / longitud del registro.
3. Crear el archivo secuencial (los registros deberán poseer la misma estructura que los del archivo directo).
4. Recuperar un registro del archivo directo haciendo referencia al número relativo de registro.
5. Grabar la información leída del archivo directo en el archivo secuencial.
6. Repetir los pasos 4 y 5 hasta encontrar el primer registro libre o hasta llegar a la cantidad de registros obtenida en el paso 2, en el caso en que todos los registros contengan datos.
7. Cerrar los dos archivos.

### *Traspaso de la información de un archivo directo a uno indexado*

1. Abrir el archivo directo.
2. Haciendo uso de una función especifica de los archivos directos, obtener la longitud del archivo. Si no se cuenta con dicha función, necesariamente se debe conocer la dimensión del archivo y longitud de cada registro para obtener la cantidad de registros por medio del cálculo: dimensión del archivo / longitud del registro.
3. Crear el archivo indexado (área de datos y área de índice), con registros que posean la misma estructura que los del archivo directo. Puede ser necesario agregar un campo para la clave.
4. Recuperar un registro del archivo directo haciendo referencia a su número relativo de registro.
5. Ubicar la nueva clave en el área de índice del archivo indexado manteniendo el ordenamiento ascendente o descendente de esta área (como clave se puede utilizar algún dato del registro del archivo directo, por ejemplo el número relativo de registro).
6. Reservar una dirección en el área de datos.
7. Grabar la información leída del archivo directo en la dirección reservada en el área de datos.
8. Asignar la dirección del área de datos reservada al índice recién creado en el área de índice.
9. Encadenar el nuevo registro a los demás registros del área de datos, si es que no están en espacios contiguos.
10. Repetir los pasos 4, 5, 6, 7, 8 y 9 hasta encontrar el primer registro libre o hasta llegar a la cantidad de registros obtenida en el paso 2, en el caso en que todos los registros contengan datos.
11. Cerrar los dos archivos.

### *Traspaso de la información de un archivo indexado a uno secuencial*

1. Abrir el archivo indexado.
2. Crear el archivo secuencial (los registros deberán poseer la misma estructura que los del archivo indexado). Puede ser necesario quitar el campo clave.
3. Recuperar un registro del archivo indexado haciendo referencia a la clave en el área de índice, con la cual se obtiene la dirección en el área de datos.
4. Grabar la información leída del archivo indexado en el archivo secuencial.
5. Repetir los pasos 3 y 4 hasta que no se encuentren más claves y direcciones en el área de índice que referencien datos en el área de datos.
6. Cerrar los dos archivos.

### *Traspaso de la información de un archivo indexado a uno directo*

1. Abrir el archivo indexado.
2. Inicializar un contador de registros (que indicará la cantidad de registros existentes en el archivo indexado).
3. Leer un registro del archivo indexado (acceder al área de índice, buscar la clave y leer la dirección; acceder al área de datos a través de esa dirección y buscar los datos asociados).
4. Actualizar el contador de registros.
5. Repetir los pasos 3 y 4 hasta que no se encuentren más claves y direcciones ene el área de índice que reverencien datos en el área de datos.
6. Cerrar el archivo indexado.
7. Dimensionar el archivo directo sabiendo que: Dimensión = contador de registros * longitud de registros. Los registros del archivo directo deben poseer la misma estructura que los del archivo indexado.
8. Abrir el archivo indexado.
9. Recuperar un registro del archivo indexado haciendo referencia a la clave en el área de índice, con la cual se obtiene la dirección en el área de datos.
10. Grabar la información leída del archivo indexado en el archivo directo, haciendo referencia al número relativo se registro, el cual se puede llegar a obtener a través de alguna función de transformación de la clave del archivo indexado (Por ejemplo: Hashing).

**11.** Repetir los pasos 9 y 10 hasta que no se encuentren más claves y direcciones en el área de índice que referencien datos en el área de datos.
**12.** Cerrar los dos archivos.

## ANALISIS ENTRE COSTO Y BENEFICIO

| | Archivo Secuencial | Archivo Indexado | Archivo Secuencial Indexado | Archivo directo |
|---|---|---|---|---|
| **Uso de la memoria** | Óptimo. | Utiliza más memoria que un archivo secuencial, pero es justificado ya que tiene un acceso bastante rápido. | Bueno. | Regular, ya que se debe dimensionar el archivo antes de crearlo, con lo que puedo llegar a malgastar memoria si sobran registros, o puede llegar a ser poca la cantidad de registros reservados. |
| **Eficiencia en búsquedas y consultas** | Es ineficiente, ya que se debe recorrer uno a uno todos los registros hasta encontrar el que se desea. El costo de las búsquedas y consultas es alto; para disminuirlo se sugiere ordenar los registros (en forma ascendente o descendente) en función de algún campo de los registros. | Son bastante eficientes, más rápido que el secuencial y que el secuencial indexado, pero más lento que el directo. | Buena, es más rápida que la secuencial, aunque más lenta que la directa. | Bastante eficiente, ya que posee costo de acceso 1. |

## TECNICA DE APAREO DE ARCHIVOS

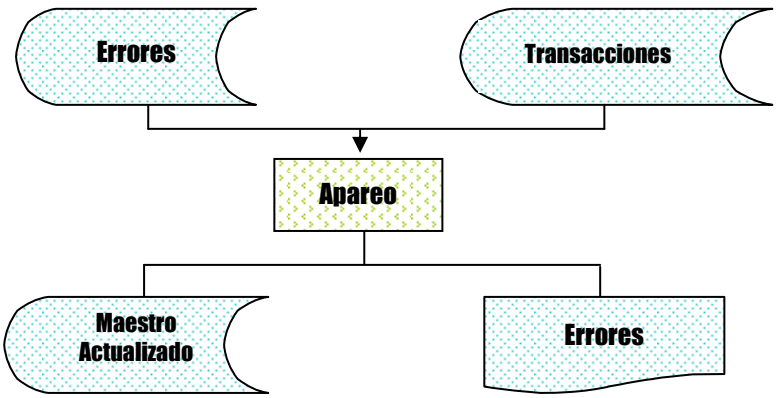
***Utilidad:*** esta técnica es utilizada para la actualización de archivos secuenciales, y consiste en confrontar dos archivos para la obtención de un resultado: informe u otro archivo.
Estos archivos mencionados se conocen como archivo maestro y archivo de transacciones.
**Archivo Maestro:** es un archivo permanente que reúne a un conjunto de registros acerca de un aspecto importante de las actividades de una organización (entidad). Por ejemplo, el archivo de empleados de una empresa, o el de artículos de un almacén, contienen datos de cada entidad existente en dicho lugar.
**Archivo de Transacciones:** es el archivo que contiene información para actualizar al archivo maestro. Puede o no tener igual estructura de registros que el maestro.
Es un archivo temporal con dos propósitos:

- Acumular datos acerca de los eventos al momento que ocurran, y
- Actualizar los archivos maestros para reflejar los resultados de las transacciones actuales.

El término transacción se refiere a cualquier evento que afecte la organización y sobre la cual se calculan datos. Los ejemplos de transacciones comunes en las organizaciones son hacer compras, pagar compras, contratar personal, pagar a los empleados y registrar ventas. Los datos importantes para la organización se reúnen en torno a cada evento y se acumulan en el archivo de transacciones.
Cada archivo de transacciones contiene únicamente los registros que pertenecen a las entidades particulares que son el tema del archivo.
Los archivos maestros son permanentes y duran mientras exista el sistema. Sin embargo, los contenidos de los archivos cambian como resultado del procesamiento y la actualización.
Para que el proceso sea posible se deben cumplir los siguientes requerimientos:

- La clave (ID) de ambos archivos debe estar compuesta por los mismos campos elementales.
- Ambos archivos deben estar ordenados por dicha clave.
- El orden de ambos archivos con respecto a la clave ha de ser el mismo.

## REPRESENTACION GRÁFICA




Esta técnica ofrece una ventaja: permite obtener un ABM y mantener ordenado al archivo maestro en una sola corrida.
La utilización de la técnica de apareo implica que cada registro en el archivo secuencial debe estar debidamente identificado, es decir, debe existir un campo o conjunto de campos que identifique unívocamente al registro.

Los errores que se pueden presentar son los siguientes:
- ***Alta duplicada:*** cuando se intenta dar de alta un registro que ya existe en el archivo maestro.
- **Baja inexistente:** cuando se intenta dar de baja un registro que no existe.
- ***Modificación inexistente:*** cuando se desea actualizar uno o más campos de un registro inexistente en el archivo.

Además, en este proceso se lleva a cabo la *validación de las transacciones*:
- Si se trata de un "alta", se tiene que tener todos los campos en el archivo de transacciones.
- Si es una "baja", el campo clave (ID) basta para esta transacción.
- Si se desea una "modificación", son suficientes en el archivo de transacciones los campos que se desean actualizar.

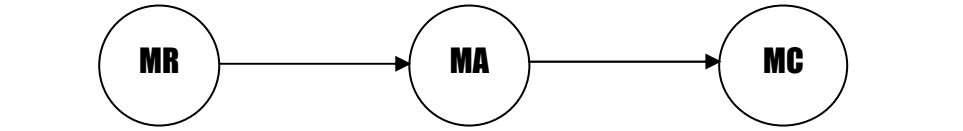
El algoritmo en Pseudocódigo para poder llevar a cabo la técnica de apareo puede tener el siguiente aspecto:

**Proceso Apareo**
Abrir <Maestro> para E;
Abrir <Transacciones> para E;
Abrir <Actualizado> para S;
Abrir <Errores> para S;
Señal:='Ambos';

```
Mientras ~FinDe (<Maestro>) ∧ ~FinDe (<Transacciones>) Hacer
    Si Señal = 'Ambos'
    Entonces  Leer (<Maestro>, Reg1);
                  Leer (<Transacciones>, Reg2);
    Sino Si Señal = 'Maestro'
            Entonces Leer (<Maestro>, Reg1);
                Sino Leer (<Transacciones>, Reg2);
            FinSi;
    FinSi ;
    Si Reg1.Id = Reg2.Id
    Entonces Si Reg2.Trans = 'ALTA'
                Entonces Escribir (<Errores>, Reg2.Trans, 'Alta duplicada') ;
                      Señal:= 'Transacciones';
                Sino Si Reg2.Trans = 'BAJA'
                          Entonces Señal:='Ambos';
                         Sino Señal:='Transacciones';
                          ModificarReg (Reg1, Reg2);
                          Escribir (<Actualizado>, Reg1);
                         FinSi;
               FinSi;
    Sino Si Reg1.Id < Reg2.Id
            Entonces Escribir (<Actualizado>, Reg1);
                  Señal:='Maestro';
            Sino Si Reg2.Trans ='ALTA';
                     Entonces Escribir (<Actualizado>, Reg2);
                    Sino Si Reg2.Trans = 'BAJA'
                              Entonces Escribir (<Errores>,Reg2.Trans, 'Baja inexistente');
                             Sino Escribir (<Errores>, Reg2.Trans, 'Modificacion inexistente');
                            FinSi;
                    FinSi;
                    Señal:= Transacciones;
            FinSi;
    FinSi;
FinMientras;
Mientras ~FinDe (<Maestro>) Hacer
   Leer (<Maestro>, Reg1);
   Escribir (<Actualizado>, Reg2);
FinMientras;
Mientras ~FinDe (<Transacciones>) Hacer
    Leer (<Transacciones >, Reg2);
   Escribir (<Actualizado>, Reg2);
FinMientras;
Cerrar <Maestro>, < Transacciones>, <Actualizado>,  <Errores>;
FinProceso.
```

# TEMA II: GRAFOS

**Teoría de grafos:** representación gráfica de un modelo de la realidad utilizando formalizaciones del álgebra. El objetivo de la teoría de grafos es de dotar de cierto rigor a determinados planteos que uno hace para hallar una solución.




**Grafos:** son dibujos que pretenden representar modelos abstractos de distintas situaciones de la realidad y con el fin último de lograr representaciones computacionales. El conjunto de puntos se representa con nodos o vértices; y las relaciones sobre ese conjunto de puntos con aristas o arcos.
Dado un conjunto P no vacío y R incluido en P x P, entonces el G (P, R) se denomina grafo dirigido en P.
**P**: conjunto de puntos, vértices o nodos que conforman la estructura.
**P**= {***x***/***x*** es un nodo del modelo}; P= {elem 1, elem 2,…, elem n}
**R**: relación que conecta a los puntos que conforman la estructura, conjunto de aristas.
**R**= {(x, y) / (x, y) ε P ^ "x se relaciona con y"}; R= {($x_1$, $y_1$), ($x_2$, $y_2$),…, ($x_n$, $y_n$)}.

En síntesis, ¿para qué sirven los grafos?: permiten representar en forma abstracta modelos de la realidad.

La forma básica de un grafo es G= (P, R), pero es posible agregar funciones de asignación resultando

**$G = (P, R, f_1, f_2,..., f_n, g_1, g_2,..., g_m)$.**

***Funciones de asignación:*** funciones adicionales al modelo que soportan información relativa al grafo –características o propiedades de los nodos o de las relaciones–. Existen dos tipos:

- ***Funciones de asignación a nodos:*** soportan las características o atributos de los nodos, caracterizan al conjunto P.
  $f_i$: P → $V_i$ donde $V_i$ representa…
- ***Funciones de asignación arcos:*** soportan las características o atributos de los arcos, caracterizan a la relación (los arcos representan las relaciones).
  $g_i$ : R → $W_i$ donde $W_i$ representa…

## CONCEPTOS

■ ***Subgrafo:*** es un grafo incluido en otro grafo. G'= (P', R') es un subgrafo de G= (P, R) si P' está incluido o es igual a P y si R' es igual a R sobre los valores de P'. En símbolos:

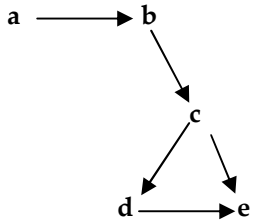
**Sea G'= (P', R') y G= (P, R); G' es subgrafo de G si $P' \subseteq P$ y $R' = R|_P$**

■ ***Paso:*** se define paso entre dos puntos cualesquiera (***x***, ***z***), como una secuencia < $y_0$, $y_1$, $y_2$,..., $y_n$ >, $n \geq 0$, que cumple con las siguientes 3 condiciones:

1. $x = y_0 \quad z = y_n$ (Todo paso tiene un origen y un final)
2. $y_{i-1} \neq y_i$ (*Secuencialidad de elementos*: los elementos consecutivos tienen que ser distintos)
3. $(y_{i-1}, y_i) \in R, \quad 1 < i \leq n$ (*Paso simple*: los elementos consecutivos deben estar a un arco de diferencia entre uno y otro)

**∂ (*x*,*z*): secuencia de puntos que partiendo desde *x* permiten llegar a *z*.**

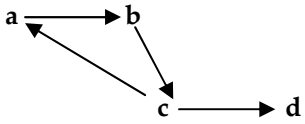
*Ejemplo:*




$|\partial (a, c)| = <a, b, c>$
$|\partial (a, e)| = <a, b, c, e>$
$= <, b, c, d, e >$

■ ***Longitud de paso:*** cantidad de arcos que hay entre el nodo origen (inicial) y el nodo destino (final) de la secuencia de puntos.
*En símbolos:* $|\partial (x, y)| =$ **# arcos entre $x$ y $z$ (es un número)**

■ ***Ciclo:*** es un paso entre un nodo y sí mismo, cuya longitud de paso es mayor que 1.
*En símbolos:* $\exists\ |\partial (x, x)| > 1$
*Ejemplo:*



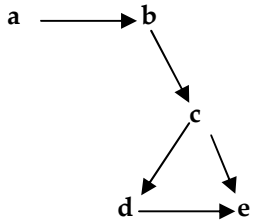

$|\partial (a, a)| = 3$

■ ***Bucle (o Loop):*** es un paso entre un nodo y sí mismo de longitud 0. Paso de longitud cero y cuyo nodo origen es igual al nodo destino.
*En símbolos:* $|\partial (x, x)| = 0$

■ ***Camino:*** un camino de $x$ a $z$ – $\omega(x, z)$ – es una secuencia de puntos en la que no interesa el sentido de la relación pero es necesario que los puntos estén relacionados (debe existir unión o conexión de puntos).
Se deben cumplir las siguientes condiciones:

1. $y_0 = x$; $y_n = z$ (Todo camino tiene un origen y un final -ídem paso-)
2. $y_{i-1} \neq y_i$ (los elementos consecutivos tienen que ser distintos -ídem paso-)
3. $(y_{i-1}, y_i) \in R \ \check{} \ (y_i, y_{i-1}) \in R$ (no importa el sentido de la relación, sino que los elementos estén relacionados).

*Ejemplo:*




$\omega (c, a) = <c, b, a>$

■ ***Camino simple:*** todos sus vértices son distintos, con excepción de $y_0$ e $y_n$, que pueden ser el mismo.

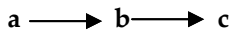
■ ***Longitud de un camino:*** es el número de arcos en ese camino.

■ ***Grafo Clausura Transitiva:*** grafo que se obtiene como resultado de inducir o generar la relación de paso sobre la relación original de un grafo G cualquiera. Es otro grafo en el cual el conjunto de puntos sigue siendo el mismo pero la relación resulta de generar todos los arcos posibles con respecto a la relación original.

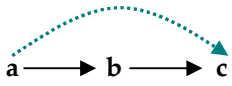
El paso es transitivo si: $\exists\, \partial\, (x, y) \wedge \exists\, \partial\, (y, z) \Rightarrow \exists\, \partial\, (x, z) \Rightarrow G^T (P, \rho_R)$

*Ejemplo:*

**G= (P, R) grafo original**




**$G^T$= (P, $P_R$)**



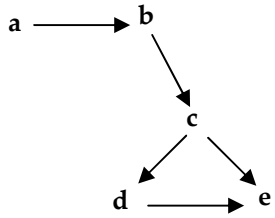

■ ***Circuito:*** es una secuencia de nodos entre $x$ y $x$ de longitud mayor o igual a dos, sin importar el sentido de la relación.

1. $|c(x, x)| \geq 2$ (longitud del camino entre $x$ y $x$ es mayor o igual que 2).
2. No importa el sentido de la relación. No respeta la relación.

■ ***Conjunto Izquierdo (o Left) de un nodo:*** es el conjunto de nodos desde los cuales se puede llegar al nodo considerado con longitud de paso igual a uno (directamente).

**$L(x)= \{ ^y/y \in P \wedge (y, x) \in a R\}$ (quien está a la izquierda de $x$)**

*Ejemplo:*

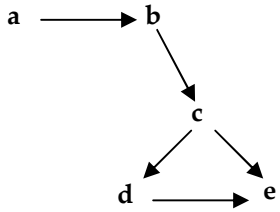



L (e) = {c, d}

■ ***Conjunto Derecho (o Right) de un nodo:*** es el conjunto de nodos a los cuales se puede llegar desde el nodo considerado con una longitud de paso igual uno (directamente).

**$R(x)= \{ ^z/z \in P \wedge (x, z) \in a R\}$ (quien está a la derecha de $x$)**

*Ejemplo:*




R (c) = {d, e}

■ ***Grado de entrada:*** cantidad de arcos que llegan a un nodo determinado. $|L(x)| \Rightarrow$ Cardinalidad del conjunto Left.

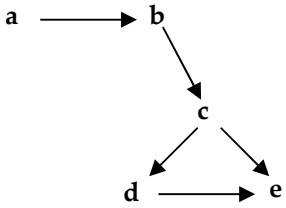
■ ***Grado de salida:*** cantidad de arcos que salen de un nodo determinado. $|R(x)| \Rightarrow$ Cardinalidad del conjunto Right.

■ ***Grado de entrada (salida) de un grafo:*** grado del nodo que posee el mayor grado de entrada (salida) del grafo.

■ ***Ideal Principal Izquierdo:*** es el conjunto de nodos desde los cuales se puede llegar al nodo considerado, directa o indirectamente.

**$\overline{L(y)}= \{ ^x/x \in P, \exists \rho (x, y) \}$**

*Ejemplo:*

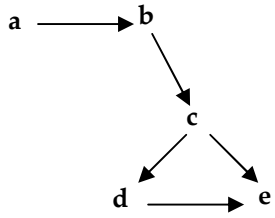



$\overline{L(d)} = \{a, b, c\}$

■ ***Ideal Principal Derecho:*** es el conjunto de nodos a los cuales puedo llegar desde el nodo considerado, directa o indirectamente.

$$\overline{R(y)} = \{ z / z \in P, \exists \rho\ (y, z) \}$$

*Ejemplo:*




$\overline{R(a)} = \{b, c, d, e\}$

$\overline{R(d)} = \{e\}$

■ ***Conjunto minimal:*** conjunto de puntos a los cuales no llega ningún arco (elementos que no tienen arcos de llegada). El grado de entrada es cero.

$$Y \text{ es minimal en } G \Leftrightarrow x \in P, \text{ si } (x, y) \in R \Rightarrow x = y$$

■ ***Conjunto maximal:*** conjunto de puntos del grafo desde los cuales no parte ningún arco (elementos que no tienen arcos de salida). El grado de salida es cero.

$$Y \text{ es maximal en } G \Leftrightarrow z \in P, \text{ si } (y, z) \in R \Rightarrow y = z$$

■ ***Mínimo:*** único punto del grafo al cual no llegan arcos. Se da cuando el conjunto minimal está formado por un único elemento (elemento que no conforma a ningún otro elemento).

$$y \text{ es mínimo} \Leftrightarrow \forall x \in P, \exists \rho\ (y, x) \wedge L(y) = \varnothing$$

■ ***Máximo:*** único punto del grafo desde el cual no salen arcos. Se da cuando el conjunto maximal está formado por un único elemento (elemento que no conforma a ningún otro elemento).

$$y \text{ es máximo} \Leftrightarrow \forall x \in P, \exists \rho\ (x, y) \wedge R(y) = \varnothing$$

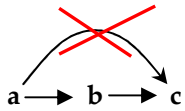
■ ***Hipergrafo:*** grafo que tiene un conjunto de puntos y más de una relación definida en dicho conjunto. Sobre el mismo modelo de la realidad es necesario analizar distintas relaciones.
Las funciones de asignación a nodos no varían porque el conjunto de puntos es el mismo, pero a las funciones de asignación a arcos se agregan aquellas características que corresponden a los arcos de las nuevas relaciones.
Esquema resultante: $H = (P, R, R', \ldots, R^{(n)}, f_1, f_2, \ldots, f_k, g_1, g_2, \ldots, g_m, h_1, h_2, \ldots, h_p, \ldots)$

■ ***Grafo finito:*** cantidad máxima de flechas que puede haber en un grafo G finito = $n^2 - n$

**Nota:** $n$ es la cantidad de nodos o puntos del modelo.

■ ***Grafo Básico:*** dado un grafo G, se dice que es básico si cumple con las siguientes condiciones:
- Está libre de bucles o loops.
- $\forall x, z \in P$, si $\exists \mid \partial(x, z) \mid \geq 2 \rightarrow (x; z) \notin R$ (no existen arcos redundantes).

Para encontrar un grafo básico hay que ir eliminando pasos que estén demás, sin destruir la estructura. Por lo general, se elimina el que tiene menor longitud de paso.
*Ejemplo:* si R= {(a, b); (b, c); (a, c)}

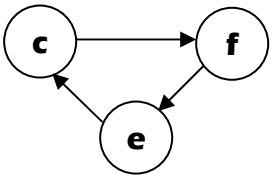



■ ***Grafo Cíclico:*** grafo que contiene por lo menos un ciclo. Puede existir más de una representación básica de él.
$\forall x \ P, \exists \mid \partial(x, x) \mid \geq 2$

■ ***Grafo Acíclico:*** grafo que no tiene ciclos. Existe una única representación básica de él.
$\forall x \ P, \nexists \mid \partial(x, x) \mid \geq 2$

■ ***Grafo Conectado o Conexo:*** grafo que no tiene puntos aislados –no tiene elementos sin relacionar–. Grafo en el que existe camino entre dos vértices cualesquiera. Un grafo es conexo si todos sus pares de vértices están conectados.

■ ***Grafo Fuertemente Conectado:*** un grafo G es fuertemente conectado si la relación de paso inducida es una relación de equivalencia. Los grafos fuertemente conectados se caracterizan porque dado cualquier par de puntos $x$ y $z$ existe un paso p($x$, $z$) que los une.




La relación de paso es una relación de equivalencia. Es reflexiva porque $\exists p$ (c, c); es simétrica porque $\exists p$ (c, f) y $\exists p$ (f, c); y es transitiva porque $\exists p$ (c, f) ^ $\exists p$ (f, e) y $\exists$ p (c, e).

■ ***Grafos Iguales:*** dos grafos son iguales sólo si uno es copia del otro.

**Si $G_1 = (P_1, R_1)$ y $G_2 = (P_2, R_2) \Rightarrow G_1 = G_2 \Leftrightarrow P_1 = P_2$ y $R_1 = R_2$**

■ ***Grafos Isomorfos:*** grafos que tienen la misma estructura pero sus nodos (componentes) son distintos.
Siendo $G_1 = (P_1, R_1)$ y $G_2 = (P_2, R_2)$, $G_1 \cong G_2 \Leftrightarrow \exists$ una función $\varphi: P_1 \rightarrow P_2$ / $\forall x, y \in P_1$, $(x,y) \in R_1 \Leftrightarrow (\varphi(x), \varphi(y)) \in R_2$ ^ $\varphi(x), \varphi(y) \in P_2$.
*Ejemplo:*

$G_1$= a ⟶ b ⟶ c
$G_2$= 1 ⟶ 2 ⟶ 3 $\qquad\qquad$ $\varphi(a)=1;\ \varphi(b)=2;\ \varphi(c)=3;$

■ ***Grafo Lineal:*** G es lineal ⇔ se verifica que:

**1.** $\exists x \in P / L(x) = \varnothing$ (hay al menos un punto al cual no le llegan arcos –minimal–)
**2.** $\forall y \in P$: $|L(y)| \leq 1$, $|R(y)| \leq 1$ (grado de entrada y grado de salida ≤ 1)
**3.** G debe ser conexo o conectado.
*Ejemplo:*

a ⟶ b ⟶ c ⟶ d es un Grafo Lineal

Un Grafo Lineal es un grafo básico que se caracteriza por ser una relación de Orden Total.

■ **Problema del camino más corto:** sea el grafo G=(P,R) en el cual cada arco tiene una etiqueta no negativa y donde un vértice se especifica como origen. El problema es determinar el costo del camino más corto del origen a todos los demás vértices y, donde la longitud de un camino es la suma de los costos de los arcos del camino. Esto se conoce con el nombre de los caminos más cortos con un solo origen. Sea G un mapa de vuelos en el cual cada vértice representa una ciudad, y cada arco $y \rightarrow x$ una ruta aérea de la ciudad $y$ a la ciudad $x$. La etiqueta del arco $y \rightarrow x$ es el tiempo que se requiere para volar de $y$ a $x$. La solución del problema de los caminos más cortos con un solo origen para este grafo determinaría el tiempo de viaje mínimo para ir de cierta ciudad a todas las demás del mapa. Para resolver este problema se puede usar una técnica que opera a partir de un conunto S de vértices cuya distancia más corta desde el orígen ya es conocida. En principio, S contiene solo el vértice de orígen. En cada paso, se agrega algún vértice restante $y$ a S, cuya distancia desde el orígen es la más corta posible. Suponiendo que todos los arcos tienen costo no negativo, siempre es posible encontrar un camino más corto entre el origen y $y$ que pasa solo a través de los vértices de S, y que se llama *especial*. En cada paso del algoritmo, se usa un arreglo D para registrar la longitud del camino especial más corto a cada vértice. Una vez que S incluye todos los vértices, todos los caminos son *especiales*, así que D contendrá la distancia más corta del orígen a cada vértice.

■ **Cerradura transitiva:** en algunos problemas podría ser interesante saber sólo si existe un camino de longitud igual o mayor que uno que vaya desde el vértice $i$ al vértice $j$. Sea que la matriz de costo $C$ es sólo la matriz de adyacencia para el grafo dado. Esto es, $C[i,j]=1$ si hay un arco de $i$ a $j$, y 0 si no lo hay. Se desea obtener la matriz $A$ tal que $A[i,j]=1$ si hay un camino de longitud igual o mayor que uno de $i$ a $j$, y 0 en otro caso. $A$ se conoce a menudo como cerradura transitiva de la matriz de adyacencia.

# TEMA III: ESTRUCTURAS LINEALES

## PSEUDOCODIGO

Sentencias mínimas necesarias:

← : asignación

If – then – else: condicional

Do – While
Do – Until
For – Next
Leer - Informar
} Repetitivas

***¿Qué es una estructura de datos?***

Se trata de un conjunto de variables de un determinado tipo agrupadas y organizadas de alguna manera para representar un comportamiento. Lo que se pretende con las estructuras de datos es facilitar un esquema lógico para manipular los datos en función del problema que haya que tratar y el algoritmo para resolverlo.

**ESTRUCTURAS LINEALES:** Conjunto de elementos de un tipo determinado relacionados entre sí.

Una estructura es lineal cuando un elemento solo puede tener un antecesor y un sucesor. Es no lineal cuando cada elemento puede tener varios antecesores y sucesores. Dos formas válidas de representación son: en memoria central y en estructuras estáticas.

***Estructuras estáticas:*** representación alternativa de las estructuras lineales. Ocupan una cantidad fija y predeterminada de memoria. Array (vectores, matrices).

Los arrays son estructuras de datos lineales, estáticas, en las que se almacena una colección de datos del mismo tipo. El almacenar sus elementos en posiciones contiguas de memoria y el ocupar un tamaño fija da lugar a dos inconvenientes: no se puede añadir ningún elemento nuevo si con anterioridad no ha sido previsto, y además es difícil borrar elementos de la lista.

Se clasifican en unidimensionales (vectores o listas) y multidimensionales (tablas matrices).

Las operaciones que se pueden realizar con arrays son lectura y escritura (agregar o quitar elementos).

Un vector es una lista de un numero finito ***n*** de elementos, que se caracteriza por almacenar los elementos en posiciones contiguas de memoria, tener un único nombre de variable que representa a todos los elementos, los que se diferencian por un subíndice, y tener acceso directo o aleatorio a los elementos.

***Estructuras dinámicas:*** el lugar que ocupan en memoria puede aumentar o disminuir en tiempo de ejecución, van ocupando y liberando memoria a medida que se necesite. Estructuras encadenadas o linkeadas.

**Pasos para crear una estructura encadenada:**

1. Reservar una celda con la cual se tiene disponible una dirección ***xx.***
2. Asignar a un puntero la dirección ***xx*** para tener direccionada esa celda.
3. Grabar la celda con los datos correspondientes (tener en cuenta que el campo de ordenamiento debe estar en nil).

**Pseudocódigo**

Rcelda (tam) → DIR
**Si** PUNT = Nil **entonces**
  Gcelda (DIR, dato, Nil)
  PUNT ← DIR (en el caso de ser una cola, a los dos punteros de acceso se les debe asignar ***Nil***)
**FinSi**

Según la inserción y eliminación de elementos, las estructuras lineales se clasifican en:

- ***Listas***
- ***Pilas***
- ***Colas***

## LISTAS

***Definición:*** forma de interrelacionar o vincular un conjunto de elementos alineados en una sola lista o fila. Sólo se necesita un vínculo por cada elemento para hacer referencia a su sucesor. Posee una operatoria dinámica abierta. No hay restricciones de por donde se insertan o se extraen las celdas.
Una lista es una estructura de datos secuencial.
Una manera de clasificarlas es por la forma de acceder al siguiente elemento:

- **Lista densa:** la propia estructura determina cual es el siguiente elemento de la lista. Ejemplo: un array.
- **Lista enlazada:** la posición del siguiente elemento de la estructura determina el elemento actual. Es necesario almacenar al menos la posición de memoria del primer elemento. Además es dinámica, es decir, su tamaño cambia durante la ejecución del programa.

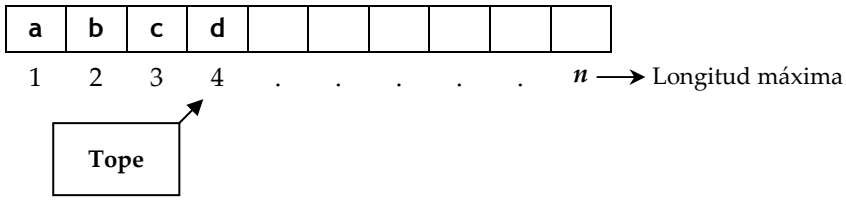
En una lista se pueden realizar las siguientes operaciones:

➔ Generación o iniciación.
➔ Inserción / eliminación de un elemento.
➔ Extracción de un elemento.
➔ Recorrido: acceder al nodo k-ésimo de la lista, para examinar y/o cambiar los contenidos de sus campos.
➔ Combinar dos o más listas lineales en una sola lista.
➔ Separar una lista lineal en una o más listas.
➔ Hacer una copia de una lista lineal.
➔ Determinar el número de nodos de una lista.
➔ Clasificar los nodos de una lista en orden ascendente, basándose en ciertos campos de los nodos.
➔ Buscar la aparición de nodos en una lista con un valor determinado en algún campo.

Una lista enlazada o encadenada es una colección de elementos ó nodos, en donde cada uno contiene datos y un enlace o liga.
Un **nodo** es una secuencia de caracteres en memoria dividida en campos (de cualquier tipo). Un nodo siempre contiene la dirección de memoria del siguiente nodo de información si éste existe.
Un **apuntador** es la dirección de memoria de un nodo.
El campo liga, que es de tipo puntero, es el que se usa para establecer la liga con el siguiente nodo de la lista. Si el nodo fuera el último, este campo recibe como valor NIL (vacío).
A continuación se muestra el esquema de una lista:


### *Implementación en estructuras estáticas*

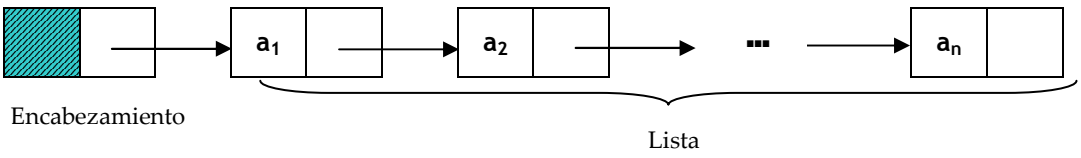
En la realización de una lista mediante arreglos, los elementos de ésta se almacenan en celdas contiguas de un arreglo. Esta representación permite recorrer con facilidad una lista y agregarle elementos nuevos al final. Pero insertar un elemento en la mitad de la lista obliga a desplazarse una posición dentro del arreglo a todos los elementos que siguen al nuevo elemento para concederle espacio. De la misma forma, la eliminación de un elemento, excepto el último, requiere desplazamientos de elementos para llenar de nuevo el vacío formado.




### *Implementación en estructuras dinámicas*

En la lista se utilizan apuntadores para enlazar elementos consecutivos. Esta implementación permite eludir el empleo de memoria contigua para almacenar una lista y, por tanto, también elude los desplazamientos de elementos para hacer inserciones o rellenar vacíos creados por la eliminación de elementos. Pero se debe pagar el precio de un espacio adicional para los apuntadores.

En esta representación, una lista está formada por celdas; cada celda contiene un elemento de la lista y un apuntador a la siguiente celda. Si la lista es $a_1, a_2, \ldots, a_n$, la celda que contiene $a_i$ tiene un apuntador a la celda que contiene a $a_{i+1}$, para $i = 1, 2, \ldots, n-1$. La celda que contiene $a_n$ posee un apuntador ***Nil***. Existe también una celda de encabezamiento que apunta a la celda que contiene $a_1$; esta celda de encabezamiento no tiene ningún elemento. En el caso de una lista vacía, el apuntador del encabezamiento es ***Nil*** y no se tienen más celdas.



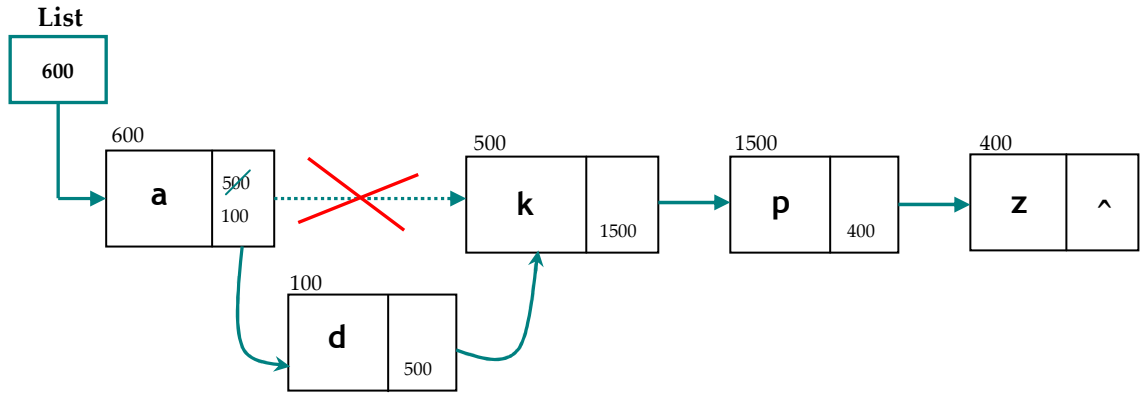

**Pueden darse distintos tipos de altas**

- **Al inicio de la lista:**
  Reservar una celda, con lo cual se tiene disponible una dirección ***yy***.
  Grabar la celda apuntada por ***yy*** con los datos correspondientes (el campo de enlace debe apuntar a la celda apuntada por el puntero de dirección).
- **Al medio de la lista** (condición: conocer la dirección de los elementos entre los cuales se debe hacer la inserción, o sea, dirección de la celda anterior y posterior a la que se quiere insertar):
  - Reservar una celda, con lo cual se tiene disponible una dirección ***yy***.
  - Leer la celda apuntada por el puntero de acceso ***List***. A una variable auxiliar (***AUX***) se le asigna el puntero ***List*** y a otra variable de tipo puntero (***RECO***) el campo de enlace de la celda leída para recorrer la estructura hasta encontrar las celdas entre las cuales debe ir el elemento. Mientras esa variable de recorrido sea distinta de ***Nil*** o no se dé el alta, se debe leer la celda apuntada por dicha variable y determinar si el elemento leído debe ser insertado en esa posición o no. Si no es la posición buscada, a ***AUX*** se le asigna el valor de la variable de recorrido y a la variable de recorrido se le asigna el campo enlace de la celda recién leída. Si es la ubicación correspondiente, entonces se procede de la siguiente manera:
  - Grabar la celda apuntada por ***yy*** con los datos correspondientes (el campo enlace debe apuntar a la celda apuntada por el elemento de recorrido).
  - Grabar la celda apuntada por el puntero auxiliar con los mismos datos apuntando a ***yy***.
  - Por último, determinar un corte indicando que el dato ya fue insertado.
- **Al final de la lista:** recorrer toda la lista hasta encontrar la marca ***Nil***.
  - Reservar espacio para la nueva celda obteniendo una dirección ***yy***.
  - Grabar la celda apuntada por ***AUX*** (puntero que se utilizó para recorrer) con los mismos datos. Pero modificando el campo de enlace, el que debe apuntar a ***yy***.
  - Grabar la celda apuntada por ***yy***, apuntando a ***Nil***.

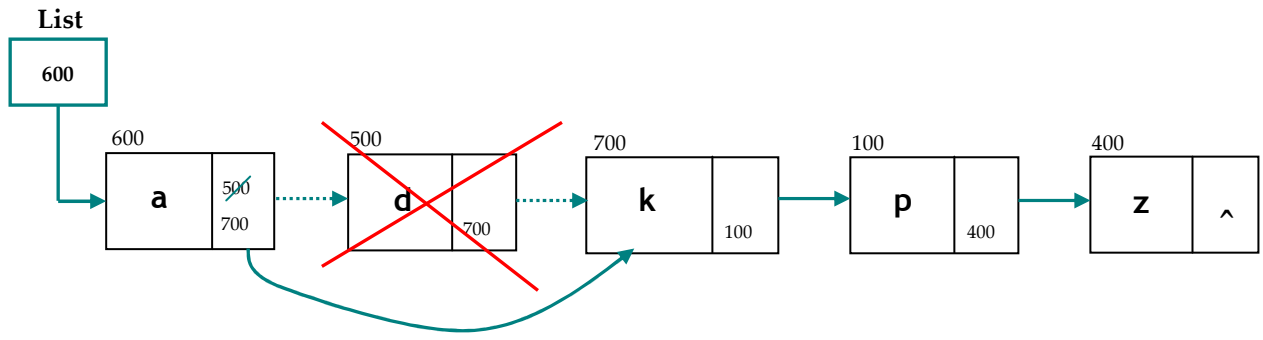
**Pseudocódigo para dar de alta en un lugar a determinar (por ejemplo con criterio descendente)**

```
Leer NUEVO (dato a ser ingresado)
BAN = 'f'
RCELDA (tam) → DIR
LCELDA (LIST, dato, enlace)
Si NUEVO < dato
   Entonces GCELDA (DIR, NUEVO, LIST)
            LIST ← DIR
   Sino AUX ← LIST
        RECO ← enlace
        Mientras RECO <> Nil y BAN = 'f' hacer
                 LCELDA (RECO, dato, enlace)
            Si NUEVO > dato
               Entonces GCELDA (DIR, NUEVO, RECO)
                        GCELDA (AUX, DIR)
                        BAN = 'v'
               Sino AUX ← RECO
                    RECO ← enlace
            Fin Si
        Fin Mientras
      Si BAN = 'f'
         Entonces GCELDA (DIR, NUEVO, Nil)
                  GCELDA (AUX, DIR)
                  BAN = 'v'
      Fin Si
Fin Si
```

***Representación Gráfica (ALTAS en estructuras dinámicas)***




***Representación Gráfica (BAJAS en estructuras dinámicas)***



***Listas doblemente enlazadas:*** Son listas donde cada elemento tiene un enlace con el elemento siguiente y con el anterior (poseen punteros en ambas direcciones).
Una lista doble, ó doblemente ligada es una colección de nodos en la cual cada nodo tiene dos punteros, uno de ellos apuntando a su predecesor (***li***) y otro a su sucesor (***ld***). Por medio de estos punteros se podrá avanzar o retroceder a través de la lista, según se tomen las direcciones de uno u otro puntero.
Una ventaja que tienen es que pueden recorrerse en ambos sentidos, ya sea para efectuar una operación con cada elemento o para insertar/actualizar y borrar. La otra ventaja es que las búsquedas son algo más rápidas, puesto que no hace falta hacer referencia al elemento anterior. Su inconveniente es que ocupan más memoria por nodo que una lista simple.
La estructura de un nodo en una lista doble es la siguiente:

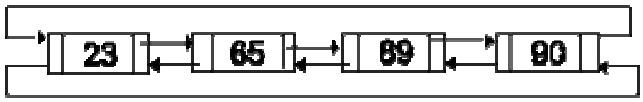

Existen dos tipos de listas doblemente ligadas:

- **Listas dobles lineales.** En este tipo de lista doble, tanto el puntero izquierdo del primer nodo como el derecho del último nodo apuntan a ***Nil***.
- **Listas dobles circulares.** En este tipo de lista doble, el puntero izquierdo del primer nodo apunta al último nodo de la lista, y el puntero derecho del último nodo apunta al primer nodo de la lista.

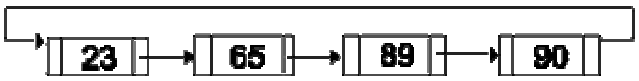
Las listas dobles circulares son más eficientes que las listas dobles lineales.

En la figura siguiente se muestra un ejemplo de una lista doblemente ligada lineal que almacena números:


En la figura siguiente se muestra un ejemplo de una lista doblemente ligada circular que almacena números:




***Listas circulares:*** las listas circulares son aquellas en las que el último elemento tiene un enlace con el primero. Es decir, tienen la capacidad de utilizar el último campo enlace, que en las listas comunes queda ocioso. En lugar de asignarle el valor nulo (***Nil***) se la hace apuntar al primer elemento de la lista. Se debe dejar referencia de cuál es la primera y cuál la última celda, para no caer en un ciclo infinito; para ello se usa un puntero de acceso que apunte a la última celda.
La siguiente figura es una representación gráfica de una lista circular.




Las listas circulares presentan las siguientes ventajas respecto de las listas enlazadas simples:

- Cada nodo es accesible desde cualquier otro nodo de la lista.
- Las operaciones de concatenación y división de listas son más eficaces con listas circulares.

Las desventajas, por el contrario, son:

- Se pueden producir lazos o bucles infinitos. Una forma de evitarlos es disponer de un nodo especial que se encuentre permanentemente asociado a la existencia de la lista circular. Este nodo se denomina ***cabecera.*** Este nodo ***cabecera*** puede tener un valor especial en su campo o una bandera que lo señale; es decir que la información del nodo cabecera no se utiliza.

***Estructura de Listas Múltiples (o Multilistas o Listas Multienlazadas):*** en general, una estructura de listas múltiples es cualquier colección de celdas, donde algunas contienen más de un apuntador y pueden, por tanto, pertenecer a más de una lista a la vez.
Para cada tipo de celda de una estructura de listas múltiples, es importante distinguir entre los campos apuntadores, de modo que se pueda seguir una lista en particular sin que haya confusión con respecto a cual de los diferentes apuntadores de una celda en particular se debe seguir.
Una estructura de datos que represente una matriz de "***n*** x ***n***" puede implementarse mediante listas múltiples con el siguiente formato de celda:

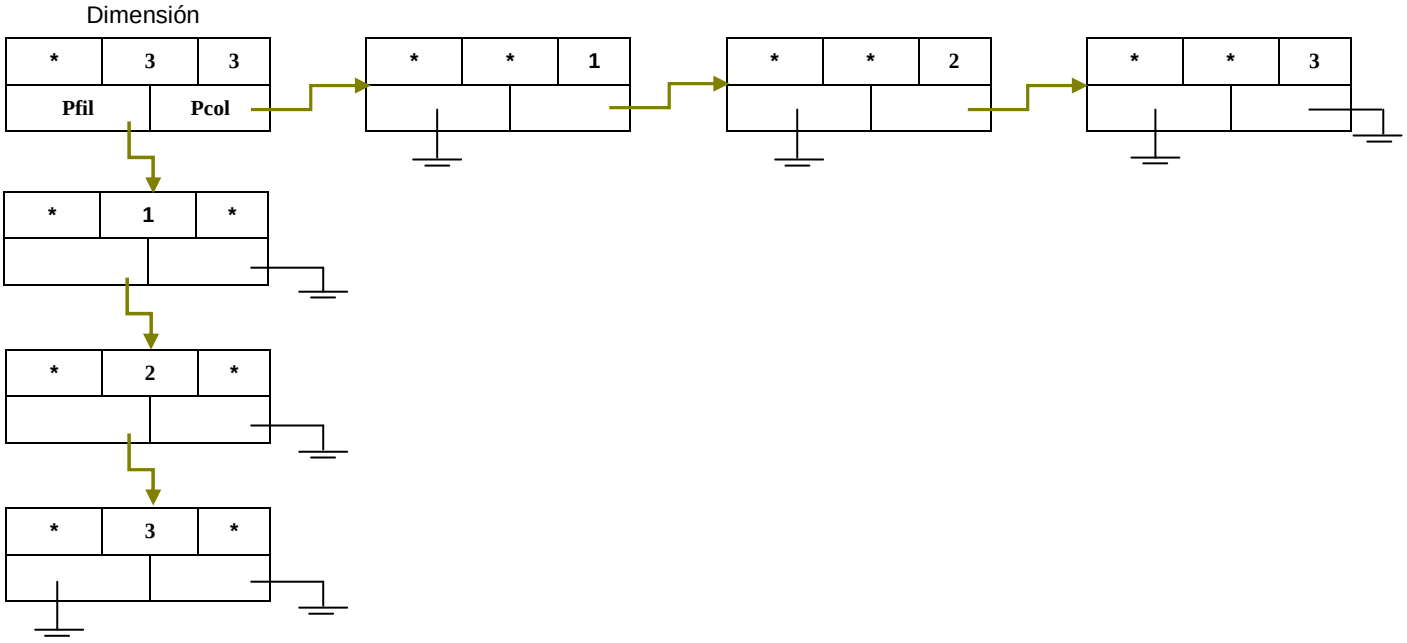
**DISEÑO DE CELDA**

| Dato | NF | NC |
|---|---|---|
| PFila | PColu | |

**REFERENCIAS**

- **NF:** número de Fila
- **NC:** número de Columna
- **PFila:** Next a Fila
- **PColu:** Next a Columna

Cada elemento de la matriz va a ser una celda. En la estructura dinámica, el puntero tendrá la cantidad de filas y columnas de la matriz. No tendrá dato. Las celdas de la primera fila no poseen datos ni el número de fila. Las celdas de la primera columna no poseen datos ni número de columna.
Usando asignaciones secuenciales de memoria, una matriz de 200 x 200 ocuparía 40000 palabras, lo que significa más memoria de la que tienen muchos ordenadores. Pero una matriz adecuada casi vacía de 200 x 200 puede representarse como la descripción anterior.
La cantidad de tiempo empleado para acceder al azar A[i, j] también es razonable, ya que sólo hay pocos elementos en cada fila y columna; y puesto que la mayoría de los algoritmos que manipulan matrices las recorren secuencialmente en lugar de acceder a elementos al azar, esta representación encadenada ocasiona poca pérdida de tiempo.
Un ejemplo de estructuras multienlazadas (se representa una matriz de 3 x 3):




Para determinar si existe paso de longitud ***n*** entre un elemento y otro se utiliza el algoritmo de WARSHALL. Dicho algoritmo consiste en ir multiplicando la matriz por sí misma para ir obteniendo en cada paso una longitud de paso mayor.

***Representación:***

$M_G = M_G^{(1)}$ → representa la matriz de paso de longitud uno.

$M_G = M_G^{(1)} * M_G^{(1)}$ → representa la matriz de paso de longitud dos.

$M_G = M_G^{(1)} * M_G^{(2)}$ → representa la matriz de paso de longitud tres.

$M_G = M_G^{(1)} * M_G^{(3)}$ → representa la matriz de paso de longitud cuatro.

……..

$M_G = M_G^{(1)} * M_G^{(n-1)}$ → representa la matriz de paso de longitud n.

Pseudocódigo:

```
For i:=1 to n do
For i:=1 to n do
For i:=1 to n do
    M [i, j]= M [i, j] + (M [i, k] * M [k, j])
```

Ejemplo: sea el siguiente grafo que representa la relación entre los números: 1, 7, 9.

**G**

$1 \rightarrow 7 \rightarrow 9$

MG

| | 1 | 7 | 9 |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 |

MG representa la relación base.

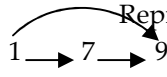
$MG^{(2)}$

| | | |
|---|---|---|
| 0 1 0 | 0 0 1 | |
| 0 0 1 | 0 0 0 | |
| 0 0 0 | 0 0 0 | |

$MG^{(2)}$ representa los pasos de longitud dos.

Del 1 al 9 hay paso de longitud

**G**




Representación de la relación existente en G

## PILA (STACK)

Las pilas son otro tipo de estructura de datos lineales, las cuales presentan restricciones en cuanto a la posición en la cual pueden realizarse las inserciones y las extracciones de elementos.

***Definición:*** es un tipo especial de lista lineal en la que la inserción y borrado de elementos se realiza sólo por uno de los extremos. Como consecuencia, los elementos de una pila serán eliminados en orden inverso al que se insertaron.

Debido al orden en que se insertan y eliminan los elementos en una pila, también se le conoce como estructura de tipo **LIFO** (Last In, First Out), es decir: último en entrar, primero en salir.

*Ejemplos:* se puede entender como una pila de libros que se amontonan de abajo hacia arriba.

En la vida cotidiana existen muchos ejemplos de pilas, una pila de platos en una alacena, una pila de latas en un supermercado, una pila de papeles sobre un escritorio, etc.

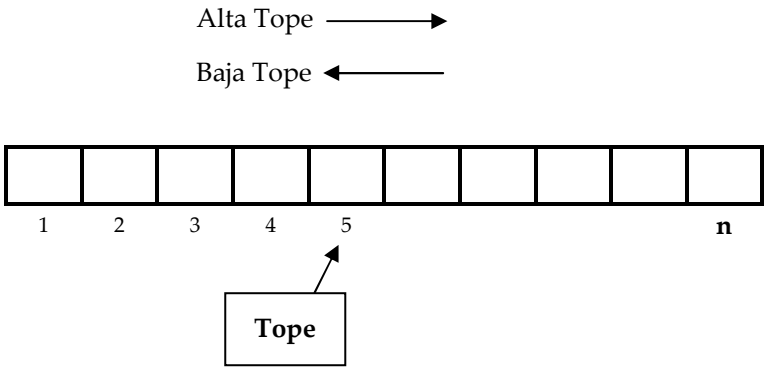
En los programas, estas estructuras suelen ser fundamentales. La recursividad se simula en un computador con la ayuda de una pila. Asimismo, muchos algoritmos emplean las pilas como estructura de datos fundamental, por ejemplo, para mantener una lista de tareas pendientes que se van acumulando.

En cuanto a su representación en memoria, se dice que las pilas no son estructuras de datos fundamentales, es decir, no están definidas como tales en los lenguajes de programación. Las pilas pueden representarse mediante el uso de:

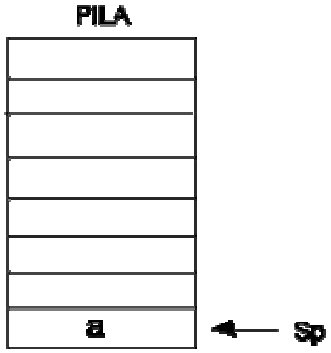
- Arreglos.
- Listas enlazadas.

### *Implementación en estructuras estáticas*

Se habla de vectores, como no hay punteros se usa un índice de vector, por ejemplo ***Tope***, que inicialmente estará cargado con 0 o con 1 y su valor variará entre 1 y N (dimensión del vector). Las altas consisten en incrementar el valor de ***Tope*** en 1. Cuando se quiere dar de baja se produce desplazamiento de ***Tope*** a la izquierda, se decrementa en 1 el valor de ***Tope***. Se debe tener en cuenta que ***Tope*** no puede ser menor a 1 (único caso pila vacía, valor inicial), ni mayor a N (si ***Tope*** = **N** → Pila llena).




Cuando se utilizan arreglos, debemos definir el tamaño máximo de la pila, además de un apuntador al último elemento insertado en la pila el cual denominaremos SP. La representación gráfica de una pila es la siguiente:




Como utilizamos arreglos para implementar pilas, tenemos la limitante de espacio de memoria reservada. Una vez establecido un máximo de capacidad para la pila, ya no es posible insertar más elementos.
Una posible solución a este problema es el uso de espacios compartidos de memoria. Supóngase que se necesitan dos pilas, cada una con un tamaño máximo de ***n*** elementos. En este caso se definirá un solo arreglo de 2****n*** elementos, en lugar que dos arreglos de ***n*** elementos.
En este caso utilizaremos dos apuntadores: SP1 para apuntar al último elemento insertado en la pila 1 y SP2 para apuntar al último elemento insertado en la pila 2. Cada una de las pilas insertará sus elementos por los extremos opuestos, es decir, la pila 1 iniciará a partir de la localidad 1 del arreglo y la pila 2 iniciará en la localidad 2***n***. De este modo si la pila 1 necesita más de ***n*** espacios (hay que recordar que a cada pila se le asignaron ***n*** localidades) y la pila 2 no tiene ocupados sus ***n*** lugares, entonces se podrán seguir insertando elementos en la pila 1 sin caer en un error de desbordamiento.

### *Implementación en estructuras dinámicas*

Se pueden realizar altas y bajas siguiendo el criterio LIFO. No se pueden realizar inserciones en el medio ni recorrer la pila, ya que la lectura de la misma es destructiva (para recorrerla se debe utilizar una estructura auxiliar).

**Pasos para dar de alta** (tener en cuenta el criterio LIFO)

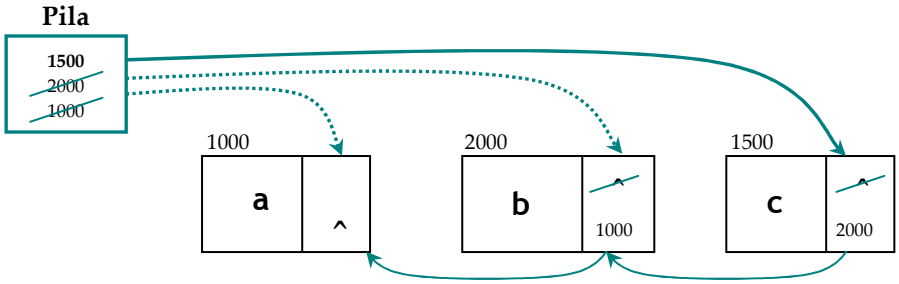
- Reservar una celda, con lo cual se tiene disponible una dirección ***yy***.
- Grabar la celda apuntada por ***yy***, con los datos correspondientes (el campo de enlace debe apuntar a la nueva celda apuntada por el puntero de acceso –***Pila***–).

× Hacer que el puntero de dirección ***Pila*** apunte a la nueva celda (se le debe asignar la dirección ***yy***).

***Pseudocódigo***

RCELDA (tam) → DIR
GCELDA (DIR, Dato, PILA)
PILA ← DIR

***Representación Gráfica (ALTAS en estructuras dinámicas)***

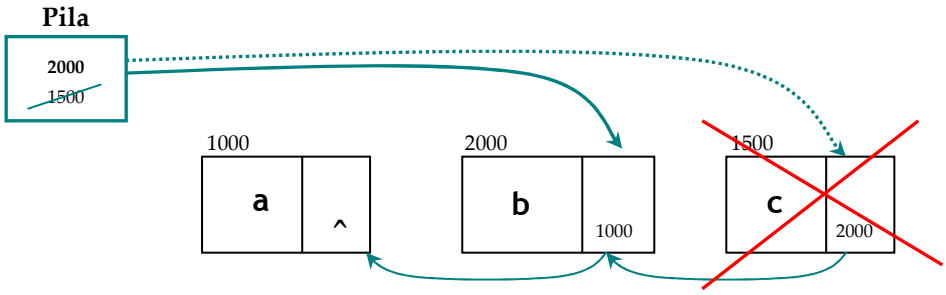



**Pasos para dar de baja:** tengo un sólo candidato a dar de baja, que es el último ingresado.
× Leer la celda apuntada por el puntero ***Pila***, para llevar los datos a memoria.
× Liberar la celda apuntada por ***Pila***.
× Cambiar la dirección del puntero ***Pila***, basándose en la dirección del campo enlace de la celda leída.

***Pseudocódigo***

LCELDA (PILA, Dato, Enlace)
LIBCELDA (PILA)
PILA ← Enlace

***Representación Gráfica (BAJAS en estructuras dinámicas)***




***Aplicaciones de las pilas: llamadas a subprogramas***

Cuando dentro de un programa se realizan llamadas a subprogramas, el programa principal debe recordar el lugar donde se hizo la llamada, de modo que pueda retornar allí cuando el subprograma se haya terminado de ejecutar.
Supongamos que tenemos tres subprogramas A, B y C, y supongamos también que A invoca a B y B invoca a C. Entonces B no terminará su trabajo hasta que C haya terminado y devuelto su control a B. De modo similar, A es el primero que arranca su ejecución pero es el último que la termina, tras la terminación y retorno de B.
Cuando un subprograma termina, debe retornar a la dirección siguiente a la instrucción que lo llamó. Cada vez que se invoca a un subprograma, la dirección de la siguiente línea (x, y o z) se introduce en la pila. El vaciado de la pila se realizará por los sucesivos retornos, decrementándose el puntero de pila que queda siempre apuntando a la siguiente dirección de retorno.

### *Elegir entre implementación con listas o con arrays*

El uso de array es idóneo cuando se conoce de antemano el número máximo de elementos que van a ser apilados y el compilador admite una región contigua de memoria para el array. En otro caso, sería más recomendable usar la implementación por listas enlazadas; también si el número de elementos llegase a ser excesivamente grande.
La implementación por array es ligeramente más rápida. En especial, es mucho más rápido a la hora de eliminar los elementos que hayan quedado en la pila. Por lista enlazada esto no es tan rápido. Por ejemplo, piénsese en un algoritmo que emplea una pila y que en algunos casos al terminar éste su ejecución deja algunos elementos sobre la pila. Si se implementa la pila mediante una lista enlazada, entonces quedarían en memoria una serie de elementos que es necesario borrar. La única manera de borrarlos es liberar todas las posiciones de memoria que le han sido asignadas a cada elemento, esto es, desapilar todos los elementos. En el caso de una implementación con array, esto no es necesario, salvo que quiera liberarse la región de memoria ocupada por éste.

## COLA (QUEUE)

Una cola es una estructura de datos de acceso restrictivo a sus elementos. Un ejemplo sencillo es una cola en el cine o del autobús, el primero que llegue será el primero en entrar.
***Definición:*** estructura de datos en la cual la operación de adición se realiza por un extremo y la de eliminación por el otro. Por el frente se extraen y por el fondo se añaden los elementos.
Es una estructura de tipo **FIFO** (First In, First Out), es decir: primero en entrar, primero en salir.

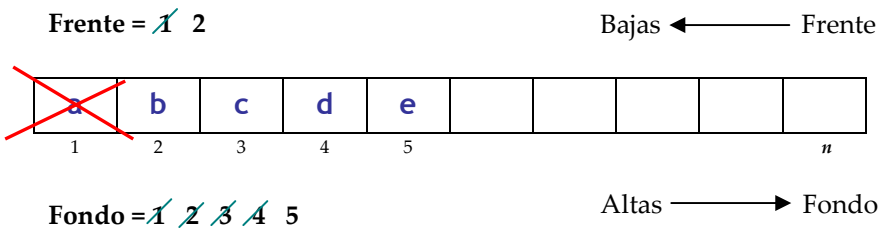
### *Operaciones básicas que se realizan sobre una cola*

- ➔ Crear o inicializar.
- ➔ Insertar un elemento.
- ➔ Determinar si una cola está vacía.
- ➔ Determinar si una cola está llena.

> **Importante:** en una cola (al igual que en una pila), no se pueden realizar **consultas**, ya que la lectura es destructiva; únicamente se pueden realizar eliminando estructuras.

### *Implementación en estructuras estáticas*

Es este caso, se necesitan dos índices de vector, ***Frente*** y ***Fondo***, para recorrer la estructura; inicialmente estarán cargados con 1 y sus valores variarán entre 1 y N (dimensión del vector). A medida que se van incorporando datos al vector, ***Fondo*** se incrementa de izquierda a derecha. Cuando se quiere dar de baja, **Frente** se incrementa de izquierda a derecha. Aunque es suficiente tener todos los índices correctamente establecidos los datos del vector siguen estando a menos que se vayan incrementando blancos o sean borrados.




### *Implementación en estructuras dinámicas*

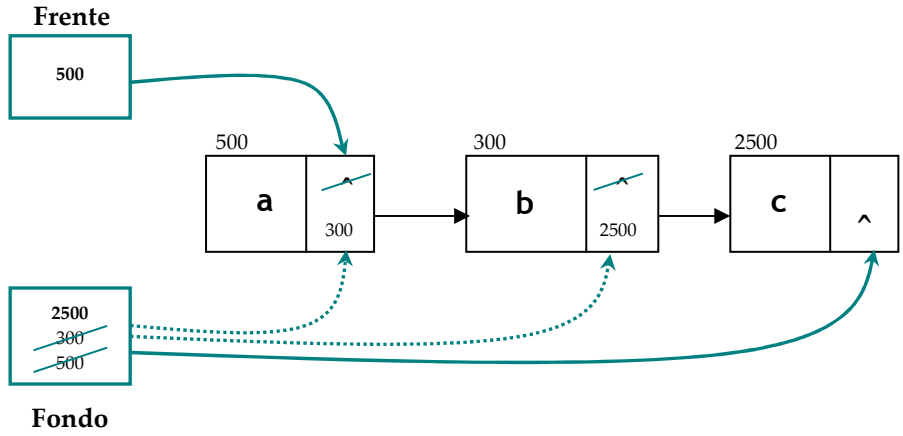
Trabaja con dos punteros, uno apunta al frente de la cola (primer elemento) y el otro al fondo (último elemento). Al crear una cola los punteros ***Frente*** y ***Fondo*** apuntan a la primera celda.
**Pasos para dar de alta** (tener en cuenta el criterio FIFO):
× Reservar espacio en memoria para la nueva celda, con lo cual se tiene disponible una dirección ***yy***.

- Grabar la celda apuntada por el puntero ***Fondo***, con los mismos datos, pero modificando el campo de enlace para hacerla apuntar a la nueva celda (o sea, la dirección ***yy***).
- Hacer que el puntero de dirección ***Fondo*** apunte a la nueva celda (se le debe asignar a ***Fondo*** la dirección de ***yy***).
- Grabar la nueva celda con los datos correspondientes, apuntando a ***Nil.***

***Pseudocódigo:***

RCELDA (tam) → DIR
GCELDA (FONDO, , DIR)
GCELDA (DIR, Dato, Nil)
FONDO ←DIR

***Representación Gráfica (ALTAS en estructuras dinámicas)***



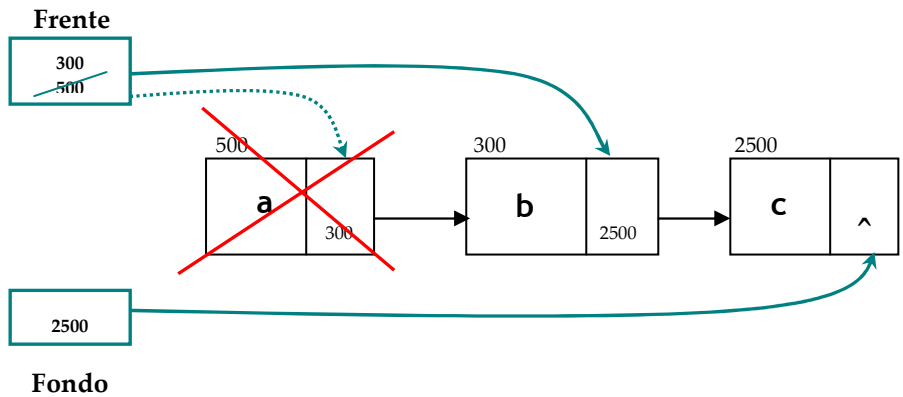

**Pasos para dar de baja**: tengo un sólo candidato a ser dado de baja, que es el primero ingresado, o sea la celda apuntada por ***Frente***.

- Leer la celda apuntada por el puntero de dirección ***Frente***, para llevar los datos a memoria.
- Liberar la celda apuntada por ***Frente***.
- Cambiar la dirección del puntero ***Frente***, basándose en la dirección del campo de enlace de la celda leída.

***Pseudocódigo:***

LCELDA (FRENTE, Dato, Enlace)
LIBCELDA (FRENTE)
FRENTE ←Enlace

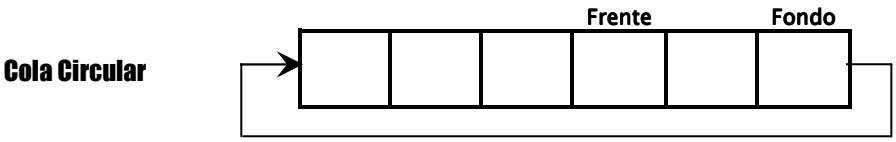
***Representación Gráfica (BAJAS en estructuras dinámicas)***



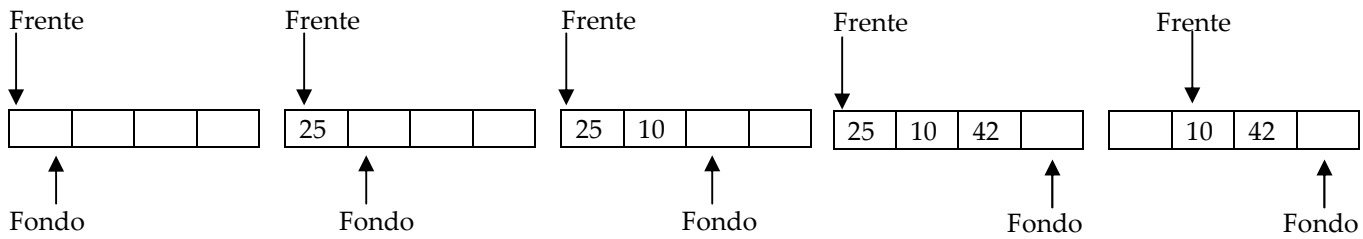
***Aplicaciones de las colas***

Un ejemplo de cola es el caso de las tareas asignadas a un microprocesador. Se utiliza una cola para almacenar los programas o las peticiones de los diferentes periféricos que esperan su turno de ejecución. El procesador atiende (normalmente) por orden de llamada, por lo tanto, todas las llamadas se almacenan en una cola. En la realidad, existe una llamada *cola de prioridades;* en ella, el micro tiene una lista de las tareas que debe ejecutar primero y sólo dentro de las peticiones de igual prioridad se producirá una cola.

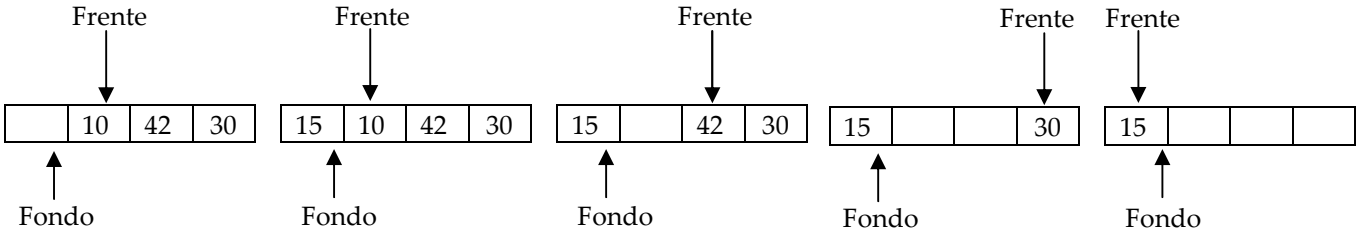
***Colas circulares:*** las colas lineales tienen un grave problema: como las extracciones sólo pueden realizarse por un extremo, puede llegar un momento en que el apuntador ***Fondo*** sea igual al máximo número de elementos en la cola, siendo que al frente de la misma existan lugares vacíos, y al insertar un nuevo elemento nos mandará un error de overflow (cola llena).
Para solucionar el problema de desperdicio de memoria se implementaron las colas circulares, en las cuales existe un apuntador desde el último elemento al primero de la cola. Éstas son colas que tienen la capacidad de reutilizar las posiciones del vector que están en desuso.
Se la maneja con dos datos, ***Frente*** y ***Fondo***. Cuando ***Fondo*** llega al final se le asigna la posición 1, siempre y cuando ***Frente*** sea distinto de 1. Si ***Frente*** = ***Fondo***, esto implica que la cola está vacía o tiene un único elemento.
Cuando ***Fondo*** está en la última posición y se quiere dar una alta, el nuevo elemento se almacenará en el primer espacio si está disponible.
La cola está vacía cuando ***Frente*** alcanza a ***Fondo***, y la cola está llena si ***Fondo*** alcanza a ***Frente.***
La cola circular se puede implementar únicamente en un arreglo estático.
**Altas:** en un principio, estas variables apuntan al mismo nodo. Cuando se realiza la primer alta, se inserta el elemento en la posición de ***Fondo***, incrementándose este puntero en uno. Para las posteriores altas se continúa con el mismo procedimiento. Cuando el ***Fondo*** llega a la última posición del vector y se da otra alta (suponiendo que hay lugar para hacerlo), el ***Fondo*** se corre al primer elemento de la lista. Finalmente, cuando ***Fondo*** alcanza la posición de ***Frente***, nos indica que la cola se ha llenado.
**Bajas:** lo que se hace es sacar el elemento apuntado por F***rente*** e incrementar en uno esta variable. Cuando la variable F***rente*** alcanza o llega a la variable ***Fondo***, nos indica que se ha vaciado la cola.
La representación gráfica de esta estructura es la siguiente:




***Ejemplo de altas y bajas en una cola circular***




La cola está vacía | Insertamos el dato 25 | Insertamos el dato 10 | Insertamos el dato 42 | Sacamos el dato 25




Insertamos el dato 30 | Insertamos el dato 15 (se llenó la cola) | Sacamos el dato 10 | Sacamos el dato 42 | Sacamos el dato 30

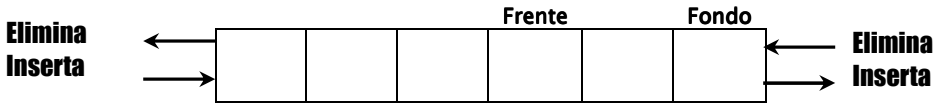
***Realización de colas con arreglos circulares***

Arreglo como un círculo en el que la primera posición sigue a la última. La cola se encuentra en alguna parte de ese círculo, ocupando posiciones consecutivas, con el extremo posterior en algún lugar a la izquierda del extremo anterior. Para insertar un elemento en la cola, se mueve el apuntador **C.post** una posición en el sentido de las manecillas del reloj, y se escribe el elemento en esa posición. Para suprimir, simplemente se mueve **C.ant** una posición en el sentido de las manecillas del reloj. De esta manera, la cola se mueve en ese mismo sentido conforme se insertan y suprimen elementos.

Un problema que existe es que no hay manera de distinguir entre una cola vacía y una cola que ocupa el círculo completo, a menos que se mantenga un bit que sea verdadero si, y sólo si, la cola está vacía. Si no se está dispuesto a mantener ese bit, se debe evitar que la cola llegue a llenar todo el arreglo.

Aún cuando el arreglo tenga **lon_max** lugares, no se puede permitir que la cola crezca más que **long_max-1**, a menos que se introduzca un mecanismo para distinguir las colas vacías.

***Doble Cola:*** Esta estructura es una cola bidimensional en que las inserciones y eliminaciones se pueden realizar en cualquiera de los dos extremos de la bicola. Sigue siendo una estructura destructible, pero permite la inserción o extracción por un mismo extremo. Gráficamente, representamos una bicola de la siguiente manera:




Existen dos variantes de la doble cola:

- Doble cola de entrada restringida.
- Doble cola de salida restringida.

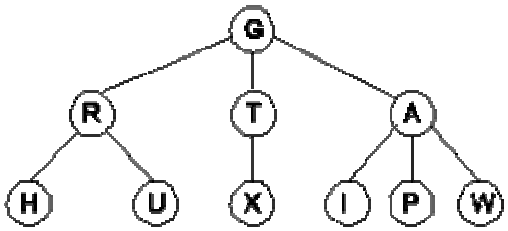
La primera variante sólo acepta inserciones al final de la cola, y la segunda acepta eliminaciones sólo al frente de la cola.

# TEMA IV: ÁRBOLES

***Introducción:*** los árboles representan las estructuras no lineales y dinámicas de datos más importantes en computación. Dinámicas porque las estructuras de árbol pueden cambiar durante la ejecución de un programa. No lineales, puesto que a cada elemento del árbol pueden seguirle varios elementos.
Los árboles pueden ser construidos con estructuras estáticas y dinámicas. Las estáticas son arreglos, registros y conjuntos, mientras que las dinámicas están representadas por listas.
Un árbol impone una estructura jerárquica sobre una colección de objetos.
Los árboles tienen una gran variedad de aplicaciones. Los árboles genealógicos y los organigramas son ejemplos comunes de árboles. Entre otras cosas, los árboles son útiles para analizar circuitos eléctricos y para representar la estructura de fórmulas matemáticas, para organizar adecuadamente la información, y para numerar los capítulos y secciones de un libro. También se presentan naturalmente en diversas áreas de computación. Por ejemplo, se usan para organizar información en sistemas de bases de datos y para representar la estructura sintáctica de un programa fuente en los compiladores.

***Definición I:*** un árbol es un conjunto finito T de uno o más nodos donde:
1. Hay un nodo especial llamado raíz del árbol, raíz (T);
2. Los nodos restantes están agrupados en $m \geq 0$ conjuntos distintos $T_1, T_2, \ldots, T_m$ y cada uno de estos conjuntos es, a su vez, un árbol. Los árboles $T_1, T_2, \ldots, T_m$ se llaman subárboles de la raíz.

La definición de árbol implica una estructura recursiva. De esto se desprende que cada nodo de un árbol es la raíz de algún subárbol contenido en la totalidad del mismo. (T= (P, R)).
Se utiliza la recursión para definir un árbol porque representa la forma más apropiada y porque además es una característica inherente de los mismos.
La figura siguiente representa a un árbol general.




***Propiedades:***
1. El grafo T verifica la existencia de un nodo $t$ el cual es mínimo (no le llega ningún arco), al cual llamamos raíz o: $\exists\, x$ *único* / $L(x) = 0$
2. Ese grafo es libre de circuitos y de loops.
3. La cantidad de nodos es la cantidad de arcos más uno: $|P| = |R| + 1$
4. El camino entre dos nodos es único (no hay arcos redundantes).

***Definición II:*** un árbol es una estructura no lineal que permite representar los datos, respetando una relación jerárquica entre ellos. Un árbol es una colección de elementos llamados nodos, uno de los cuales se distingue como raíz, junto con una relación (de paternidad) que impone una estructura jerárquica sobre los nodos. Un solo nodo es, por sí mismo, un árbol. Ese nodo es también la raíz de dicho árbol.

***Definición III:*** es una estructura jerárquica aplicada sobre una colección de elementos u objetos llamados nodos; uno de los cuales es conocido como raíz. Además se crea una relación o parentesco entre los nodos dando lugar a términos como padre, hijo, hermano, antecesor, sucesor, ancestro, etc. Formalmente, se define un árbol de tipo T como una estructura homogénea que es la concatenación de un elemento de tipo T junto con un número finito de árboles disjuntos, llamados subárboles. Una forma particular de árbol puede ser la estructura vacía.

***Definición IV:*** un árbol es un grafo G (P, R) finito y conectado tal que |P|= |R|+ 1.
G (P, R) es un árbol si y sólo si cumple al menos una de estas tres condiciones:

1. G debe ser conectado y no existe |c (***x***, ***x***)| > 1.
2. G libre de circuitos y cualquier arco adicional crea circuito.
3. G libre de circuitos y se cumple que |P|= |R|+ 1.

***Propiedades***

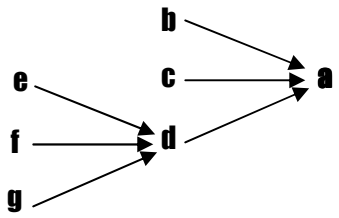
1. No existen loops.
2. ∃ c (***x***, ***y***) y es único.
3. Cada subárbol es en sí mismo un árbol.
4. Todo camino en un árbol es simple.
5. Si ***x*** ∈ P, ***x*** ∈ minimal, L(***x***)=0.
6. Si ***z*** ∈ P, ***x*** ∈ maximal, R(***z***)=0.

Existen básicamente dos tipos de árboles:

**Árbol Principal Izquierdo (API):** ***<estructura jerárquica de tipo ascendente>***

***Definición:*** un grafo G es un Árbol Principal Izquierdo si y sólo si cumple las siguientes dos condiciones:

1. Existe ***x*** perteneciente a ***P*** que es máximo. ∃ ***x*** ∈ P / ***x*** es máximo.
2. El grado de salida de un punto cualquiera salvo la raíz es 1. |R(***y***)| = 1; ∀ ***y*** ≠ ***x***, ∀ ***y*** ∈ P

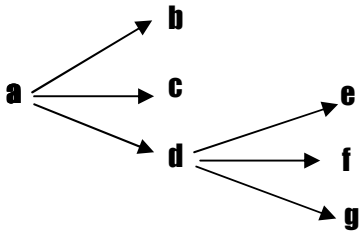



*Características*: existen distintos nodos de acceso, por lo tanto debe existir un puntero para cada uno (para cada elemento minimal), para poder recorrerlos a todos.
*Desventajas:* el API es poco aplicable puesto que posee muchas raíces (accesos, comienzos). Puede haber varios nodos de acceso y no se sabe por donde empezar. Con ello, cuanto más grande es el árbol, la estructura es ineficiente.

**Árbol Principal Derecho (APD):** ***<estructura jerárquica de tipo descendente>***

***Definición:*** un grafo G es un Árbol Principal Derecho si y sólo si cumple las siguientes dos condiciones:

1. Existe ***x*** perteneciente a ***P*** que es mínimo. ∃ ***x*** ∈ P / ***x*** es mínimo (raíz).
2. El grado de entrada de un punto cualquiera salvo la raíz es 1. |L(***y***)| = 1; ∀ ***y*** ≠ ***x***, ∀ ***y*** ∈ P




*Características*: de un nodo, se puede llegar a todos los demás. Existe un solo nodo de acceso. Con un único puntero (al elemento mínimo) se puede acceder a todos los elementos del árbol. Todo árbol con un elemento mínimo es principal derecho y, consecuentemente, todo elemento del árbol es alcanzable desde este punto mínimo. Cualquier implementación de árbol con un único puntero de ingreso a la estructura que apunte al elemento mínimo es accesible completamente (para recorrer todo el árbol, es suficiente con tener acceso a la raíz).
*Ventajas:* es mejor. Su estructura de datos es muy eficiente.

*Desventajas:* los algoritmos son más complicados.

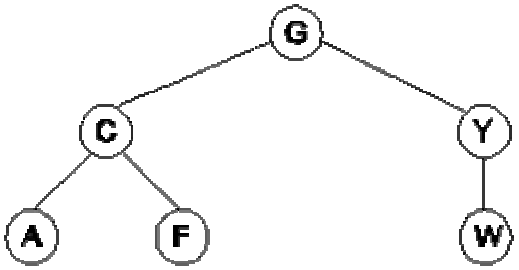
## NOMENCLATURA SOBRE ÁRBOLES

- ***Raíz:*** nodo superior de un árbol. No forma parte de ningún subárbol del árbol.
  Es aquel elemento que no tiene antecesor.
- ***Rama:*** cada subárbol que posee un nodo del árbol. Arista entre dos nodos.
- ***Hoja:*** nodos que no disponen de ningún subárbol (elementos terminales).
- ***Nodo interior:*** nodo que posee algún subárbol.
- ***Nodo descendiente:*** nodo situado directamente debajo de otro. Un nodo sin descendientes propios se denomina hoja.
- ***Sucesor:*** un nodo $x$ es un sucesor de un nodo $y$ si por algunas de las ramas de $y$ se puede llegar a $x$.
- ***Nodo ascendiente:*** nodo situado directamente encima de otro. En un árbol, la raíz es el único nodo que no posee antecesor.
- ***Antecesor:*** un nodo $x$ es un antecesor de un nodo $y$ si por algunas de las ramas de $x$ se puede llegar a $y$.
- ***Niveles:*** distintas distancias de las cuales me voy alejando de la raíz (la cual está en el nivel cero o nivel más alto). El nivel de un árbol es la longitud de paso desde la raíz hasta el nodo que se encuentra en la rama mayor de un árbol. El mayor nivel coincide con la altura de un árbol. Número de ramas que hay que recorrer para llegar de la raíz a un nodo.
- ***Altura o profundidad:*** es la mayor longitud de paso entre la raíz y sus hojas. Es la distancia desde la raíz hasta el nodo terminal más alejado. Número de niveles que posee el árbol. Máxima cantidad de arcos desde la raíz hasta un nodo terminal. Paso máximo desde la raíz hasta un nodo terminal. A la raíz se le asignará profundidad 1. La altura es el nivel más alto del árbol.
- ***Ancho de un árbol:*** máxima cantidad de nodos en un mismo nivel del árbol.
- ***Grado de un nodo:*** el número de descendientes directos que tiene.
- ***Grado o cardinalidad de un árbol:*** está dado por el mayor grado de salida de sus nodos. Número máximo de subárboles que puede tener un nodo. Cantidad máxima de arcos que salen de un nodo. El grado de un árbol determina su nombre.
  **Árbol binario:** mayor grado de salida = 2, $|R(x)| \leq 2$.
  **Árbol ternario:** mayor grado de salida = 3, $|R(x)| \leq 3$.
  **Árbol n-ario:** mayor grado de salida = $n$, $|R(x)| \leq n$.

♦ **Nota:** en el diseño es necesario tener tantos campos de enlace como lo indica el grado de salida del árbol. Si tenemos un árbol de grado $n$, se definen $n$ enlaces. La representación será eficiente si el árbol a representar es un árbol lleno, sino resulta muy caro o costoso en términos de memoria, dado que no se ocuparán todos los campos definidos.
Si la representación computacional se implementa mediante arrays, como es una estructura estática y debe dimensionarse, la primera condición a cumplir para implementar un árbol en esta estructura es conocer su altura y grado.

## Árbol binario:

***Definición I:*** es un árbol de grado 2 en el que cada nodo: no tiene hijos, tiene un hijo izquierdo, un hijo derecho, o un hijo izquierdo y un hijo derecho.
Las aplicaciones de los árboles binarios son muy variadas, ya que se los puede utilizar para representar una estructura en la cual es posible tomar decisiones con dos opciones en distintos puntos.
Un ejemplo típico de árbol binario es la representación de las eliminatorias de un campeonato de fútbol. En cada nodo estaría un ganador del enfrentamiento entre los equipos, cuyos nodos son apuntados por dicho nodo.
La representación gráfica de un árbol binario es la siguiente:

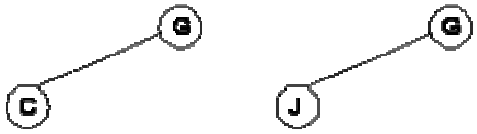



## Clasificación de Árboles Binarios

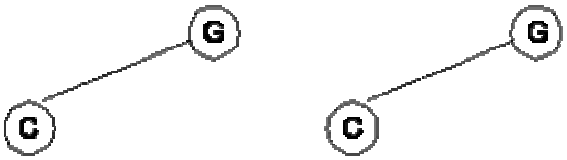
Existen cuatro tipos de árbol binario:

- **A. B. DISTINTOS**: Se dice que dos árboles binarios son distintos cuando sus estructuras son diferentes. *Ejemplo:*


- **A. B. SIMILARES:** Dos árboles binarios son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos es diferente. *Ejemplo:*




- **A. B. EQUIVALENTES:** Son aquellos árboles que son similares y que además los nodos contienen la misma información. *Ejemplo:*




- **A. B. COMPLETOS:** Son aquellos árboles en los que todos sus nodos, excepto los del ultimo nivel, tienen dos hijos; el subárbol izquierdo y el subárbol derecho.

## Árboles libres

***Definición:*** un grafo G es un árbol libre sí y sólo si cumple con alguna de estas condiciones:

- G es conectado y libre de circuitos.
- G es libre de circuitos y cualquier arco adicional genera circuitos.
- G es libre de circuitos y se cumple que la cantidad de puntos es igual a la cantidad de arcos más uno. (|P|= |R|+1)

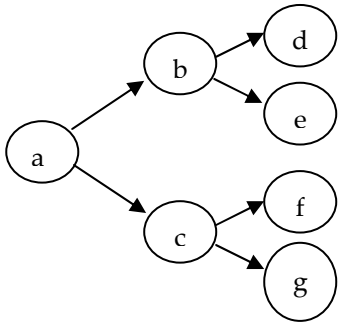
**Árbol completo:** es aquel árbol en que todos los nodos tienen el mismo grado de salida, excepto en las hojas. O sea, para todo $y$ perteneciente al conjunto de puntos y no así al conjunto maximal (no es una hoja terminal), se debe verificar que el grado de salida sea el mismo. Todos los nodos del árbol poseen la misma cantidad de hijos (grado de salida); dicha cantidad coincide con la cardinalidad.

***En símbolos:***

$\forall\ y \in P,\ y \neq \text{maximal} \Rightarrow |R(y)| = n$

**Árbol lleno:** un árbol es lleno cuando es completo y además los nodos maximales (hojas) están en un mismo nivel. La distancia desde la raíz hacia todos los nodos terminales es la misma; los nodos maximales tienen la altura del árbol.

*Ejemplo:*



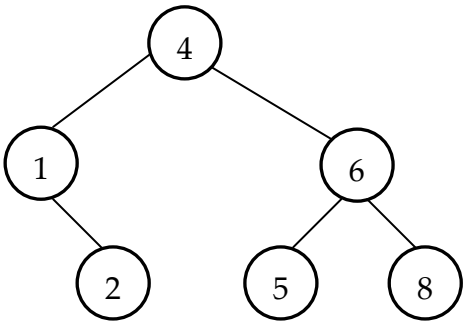

## Árboles Binarios de Búsqueda o Lexicográficos

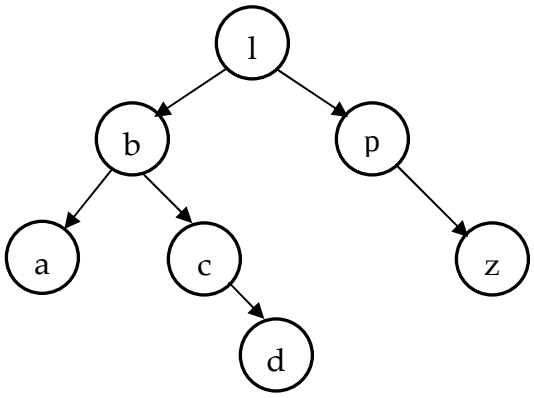
Es un caso particular de árbol donde sus elementos son dados de alta en forma ordenada. Es un árbol binario que se crea respetando el siguiente criterio: a medida que se van ingresando los elementos (altas), se van colocando los menores o iguales a la izquierda y los mayores a la derecha de la raíz.

Un árbol binario de búsqueda es aquel en el que el hijo de la izquierda (si existe) de cualquier nodo contiene un valor más pequeño que el nodo padre, y el hijo de la derecha (si existe) contiene un valor más grande que el nodo padre.

La propiedad del árbol binario de búsqueda hace que sea simple la prueba de pertenencia de un elemento. Para determinar si ***x*** pertenece, primero se compara ***x*** con la raíz. Si ***x*** es igual que el elemento de la raíz, entonces se confirma la pertenencia. Si ***x*** < raíz, entonces ***x***, si existe, sólo puede ser un descendiente del hijo izquierdo de la raíz. De igual modo, si ***x*** > raíz, ***x*** solo puede ser un descendiente derecho de la raíz.

*Ejemplos:*




Secuencia <l, b, p, c, d, a, z>

***Subárbol:*** porción de un árbol tomada a partir de un nodo que se asume como raíz y que satisface las condiciones de la definición de árbol. Una hoja o nodo terminal puede ser considerado subárbol.

## BARRIDO DE ÁRBOLES

Es un algoritmo que devuelve la secuencia de todos los nodos contenidos en un árbol. Todo árbol considerado como estructura es accedido por la raíz.

Hay cuatro criterios distintos para el barrido de árboles, tres de los cuales son aplicables sólo en árboles binarios, el cuarto barrido (por niveles) es aplicable a cualquier árbol: *preorden, postorden, simétrico y por niveles.*

Los recorridos de un árbol en los órdenes previo y posterior permiten determinar los antecesores de un nodo.

- ***BARRIDO PREORDEN (RID):*** primero se incorpora a la secuencia la raíz, luego la rama izquierda y por último la derecha. Se agotan las instancias por cada subárbol que se considera.

*Pseudocodigo:*

```
S ← ^
S ← Raíz
While S ≠ ^ do
    p ← S                          // S: pila
    visitar p //mostrar
    if R(p) ≠ ^
        then S ← R(p)
    endif
    if L(p) ≠ ^
        then S ← L(p)
    endif
```

- ***BARRIDO POSTORDEN (IDR):*** primero se incorpora a la secuencia la rama izquierda, luego la derecha y por último la raíz.

*Pseudocodigo:*

```
S ← ^
S ← (Raíz,1)
While S ≠ ^ do
    (p,i) ← S
    CASE
      i=1  S ← (p,2)
           if L(p) ≠ ^
               then S ← (L(p),1)
           endif
      i=2  S ← (p,3)
           if R(p) ≠ ^
               then S ← (R(p), 1)
           endif
      i=3  visitar p // mostrar
```

- ***BARRIDO SIMETRICO (IDR):***

**Creciente (IRD):** primero se incorpora a la secuencia la rama izquierda, luego la raíz y por último la rama derecha. Se entra en profundidad por la rama izquierda y se llega al nodo terminal.
**Decreciente (DRI):** primero se incorpora a la secuencia la rama derecha, luego la raíz y por último la rama izquierda. Se entra en profundidad por la rama derecha y se llega al nodo terminal.

*Pseudocodigo:*

```
S ← ^
S ← (Raíz,1)
While S ≠ ^ do
    (p, i) ← S
    if i=1
        then S ← (p, 2)
      if L(p) ≠ ^
        then S ← (L(p), 1)
      end if
      else visitar p // mostrar p
        if R(p) ≠ ^
            then S ← (R(p), 1)
        end if
    end if
```

- ***BARRIDO POR NIVELES:*** se obtiene la secuencia en orden ascendente por niveles y dentro de cada nivel, de izquierda a derecha. Se avanza por niveles, primero el 0, luego el 1, y así hasta el último nivel.

*Pseudocodigo:*

```
c ← ^
c ← Raíz
While c ≠ ^ do
    p ← c                          // c: cola
    visitar p //mostrar
    if L(p) ≠ ^
        then  c ← L(p)
    end if
    if R(p) ≠ ^
        then  c ← R(p)
    end if
```

**Como transformar un árbol n-ario en uno binario:** todas las estructuras de árbol n-ario pueden llevarse a binario para poder trabajar con ellas, utilizando el algoritmo de transformación denominado "Transformada de Knuth".
Si tenemos un grafo G= (P, R) que es representativo de un árbol n-ario, como resultado de la transformación se obtiene otro grafo G'= (P, $R_1$, $R_2$) –hipergrafo– que es representativo de un árbol binario.
La raíz del árbol binario es la misma que la del árbol n-ario.
$R_1$ implementa la relación existente entre un nodo y el primer elemento de su conjunto de salida. Relación de generación de la rama izquierda.
$R_2$ implementa la relación existente entre un nodo y los restantes elementos de su conjunto de salida. Arma la rama derecha en base al resto del conjunto de salida.

**Como transformar un árbol binario en uno n-ario (antitransformada de Knuth):** si se tiene un árbol binario que es resultado de haber aplicado la transformada de Knuth, se lo puede convertir en el n-ario original. La raíz del n-ario será la misma del binario, la primera relación encadena el primer hijo hacia la izquierda de la raíz del árbol n-ario, este nodo encadena hacia la derecha (en el árbol binario) a sus hermanos de nivel en el árbol n-ario.

♦ **Nota:** Hacer un barrido Preorden sobre un árbol n-ario es lo mismo que hacer un barrido Preorden sobre un árbol binario armado con el criterio de transformación.
Hacer un barrido Postorden sobre un árbol n-ario es lo mismo que hacer un barrido Simétrico sobre un árbol binario armado con el criterio de transformación.

## Árboles Balanceados

Un árbol está equilibrado en el sentido en que el número de nodos del subárbol izquierdo difiere a lo más en una unidad del número de nodos del subárbol derecho.
Los árboles balanceados presentan la mitad de sus nodos en el subárbol izquierdo y la otra mitad en el subárbol derecho.
La ventaja reportada por un árbol balanceado es evidente, ya que se aprovechan al máximo los punteros de los nodos que apuntan a los subárboles. Además, si el proceso de búsqueda dentro de un árbol es importante, cada vez que sea preciso decidir si un elemento está en alguno de los subárboles quedará reducido a la mitad el número de elementos a buscar.
Si se conoce a priori el número de nodos a insertar en el árbol, es fácil diseñar un algoritmo para construir este tipo de árboles, pero si además se desea construir un árbol balanceado ordenado que soporte las operaciones de inserción y eliminación, el esfuerzo para mantener el árbol balanceado puede ser muy alto. Por otra parte, si se permite que el árbol no esté balanceado, puede ocurrir que al construirlo degenere en una simple lista lineal, con la consiguiente pérdida en la eficacia de la operación de búsqueda.

La mayoría son árboles de búsqueda (binarios o n-arios). Los árboles llenos y completos (cuyas ramas no difieran en más de un nivel) son árboles balanceados. Al incorporar un nuevo elemento $x$ se debe mantener el balance creando nodos ficticios o reacomodar los nodos, lo que resulta un inconveniente grande, por lo que surgen los árboles AVL.

## Árboles AVL (Adel'son-Vel'skii-Landis)

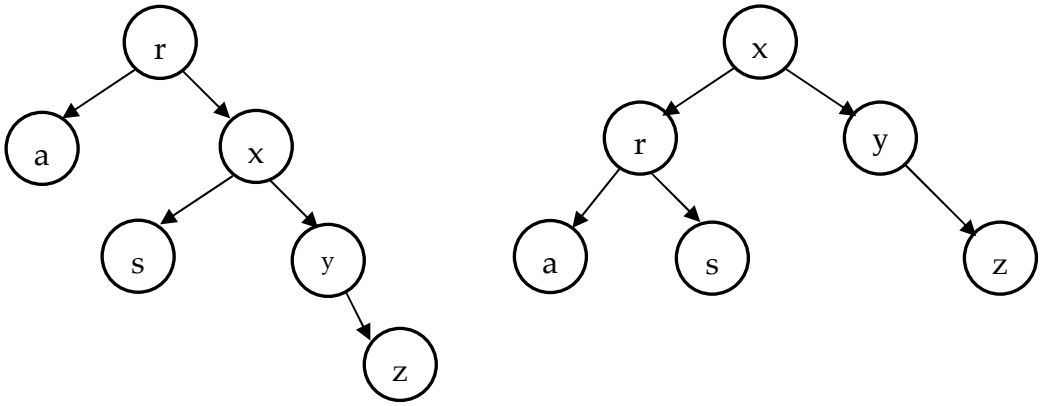
***Definición:*** son árboles binarios de búsqueda en el cual las alturas de los subárboles derecho e izquierdo de la raíz difieren en no más de una unidad (una unidad como máximo), y además sus subárboles son a su vez AVL.

Los árboles AVL tratan de generar árboles ordenados cuya estructura se asemeje a la de un árbol balanceado, pero de tal forma que las operaciones para reequilibrarlo, tras una operación de inserción o eliminación, sean sencillas.
Son ejemplos de árboles AVL un nodo (altura 0, cumple con la propiedad), un árbol con una sola descendencia (hijo), un árbol en el que la altura del subárbol derecho difiere en 1 de la altura del subárbol izquierdo y una árbol en el que la altura del subárbol izquierdo difiere en 1 de la altura del subárbol derecho.
***Criterio de equilibrio:*** un árbol está equilibrado siempre que las alturas de los subárboles de todos sus nodos difieran a lo más en una unidad.
Este requisito es menos restrictivo que el correspondiente a un árbol balanceado; de todas formas, todos los árboles balanceados son árboles AVL, pero estos últimos mantienen una estructura más que razonable, permitiendo además realizar operaciones relativamente sencillas sobre ellos sin que por ello pierdan el equilibrio AVL.
***Factor de balance:*** elemento adicional que se usa en cada uno de los nodos para determinar si se mantiene o no la condición de AVL. Concepto que define el balance de la carga de cada nodo. Existen tres estados posibles: balanceado, desbalanceado con carga a la derecha y desbalanceado con carga a la izquierda.
Cuando un árbol está perfectamente balanceado, el factor de balance se indica con un guión. Los nodos terminales siempre tienen factor de balance perfecto ya que sus descendientes son nulos (factor de balance igual). Los nodos que tengan desbalance se denotan con una barra inclinada hacia la derecha o la izquierda, dependiendo de la dirección de mayor altura (factor de balance derecho alto o izquierdo alto). En el diseño de celdas, se añade un campo más donde colocar la marca de balance.

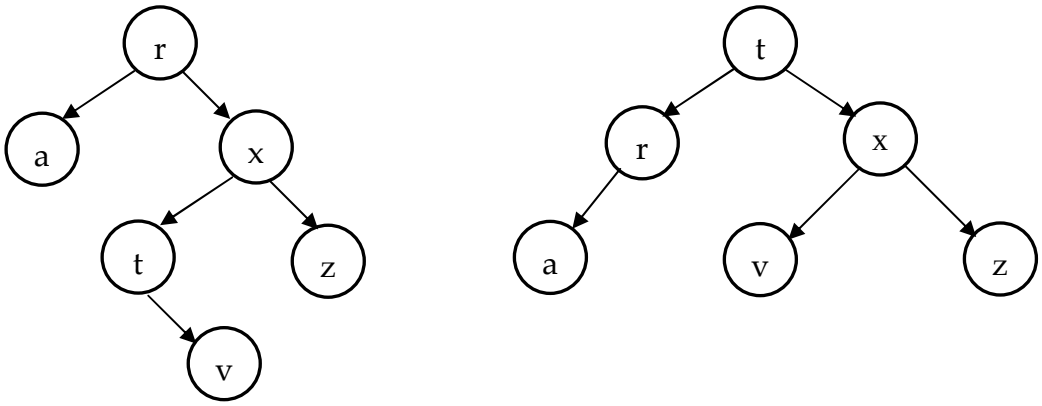
### *Inserción*

- Podemos insertar un nuevo nodo en un árbol AVL utilizando el algoritmo usual del árbol binario de búsqueda.
- A menudo resulta que el nuevo nodo puede insertarse sin cambiar la altura del árbol. Si al insertar un nodo la altura de un árbol no cambia con respecto a la raíz, el árbol sigue siendo AVL, el balance de la raíz no se modifica.
- Puede ocurrir que la altura de un subárbol aumente pero afectando la rama más corta, por lo tanto el árbol sigue siendo AVL, aunque el factor de balance de la raíz cambie (el factor de balance del nodo padre siempre cambia cuando se realiza una inserción).
- El único caso que puede ocasionar problemas se presenta cuando el nuevo nodo se agrega al subárbol más alto de la raíz, por lo tanto al aumentar la altura de este subárbol, ahora un subárbol tendrá un altura mayor en dos que el otro y ya no satisfacerá los requisitos de AVL. En tal caso, se debe reestructurar el árbol para reestablecer el balanceo haciendo uso se una rotación que consiste en tratar de solucionar el desbalance en el subárbol en el que se produjo el exceso; si no se puede solucionar en ese nivel entonces se sube un nivel más (el desbalance se debe corregir aplicando una rotación adecuada, lo más cerca posible del alta). En síntesis, la rotación de un AVL es una reestructuración del árbol luego de una inserción o una eliminación que produce un desbalance que hace perder la condición de AVL. La rotación puede ser simple o doble.

- ✓ **Rotación Simple:** el exceso en profundidad se da en un sentido, el crecimiento es a izquierda o a derecha. La raíz del subárbol donde se produjo el desbalance sube un nivel, y a partir de aquí el árbol se reestructura como un árbol binario de búsqueda.
  ***Ejemplo:***




- ✓ **Rotación doble:** se da para el caso en el que el desbalance no se produce en un sentido único. La raíz del subárbol en el que se produjo el exceso sube dos niveles, se efectúan dos rotaciones.
  ***Ejemplo:***




### *Bajas*

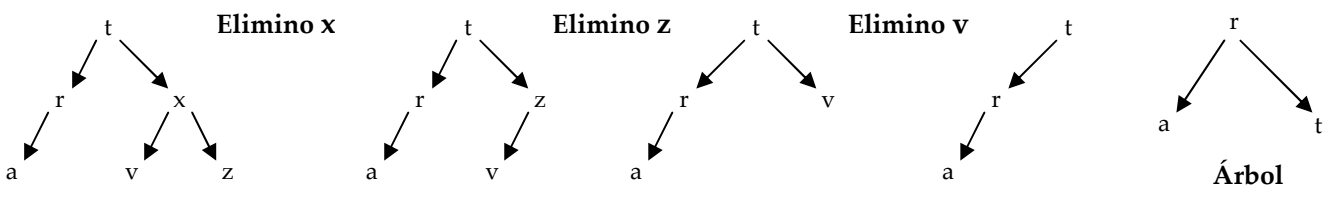
Para suprimir un nodo en un árbol AVL se utilizan las mismas ideas básicas que en la inserción.
Existen dos casos (en ambos casos si se pierden las condiciones de AVL debe aplicarse la rotación que corresponda para balancearlo):

**Caso 1:**

Es aquel en el cual el nodo a dar de baja es un nodo cualquiera del árbol AVL, que no sea la raíz.

- Si tiene dos hijos, se sube como raíz el hijo de la rama derecha.
- Si el nodo que se quiere eliminar tiene un solo hijo, obviamente el hijo sube al lugar del padre (como raíz).
- Si no tiene hijos, directamente se lo elimina.

Ejemplo:



**Caso 2:**
Eliminar la raíz del árbol. Hacer esto implica que se puede pasar como raíz la raíz del subárbol izquierdo o del derecho.

## Árboles Multimodales o de Búsqueda Multimodal

Son aquellos árboles que se generalizan de los árboles binarios de búsqueda. Sus características principales son:

- Todo árbol multimodal tiene un orden llamado ***m***.
- Cada nodo tendrá como máximo ***m*** hijos. Además, si $k <= m$, el nodo tendrá ***k*** - 1 claves, que divide todas las claves en ***k*** subconjuntos.
- Si alguno de estos subconjuntos está vacío, los hijos de estos subconjuntos también estarán vacíos. Dependiendo de la cantidad de claves se asocia cuantos hijos tendrá en cada elemento (será uno más).

## Árboles B (Bayer)

***Definición:*** Una estructura se considera Árbol B cuando es una estructura multimodal de orden '***m***' que cumple con las siguientes condiciones:

- Todas las hojas (nodos terminales) están en el mismo nivel. Se puede decir que está equilibrado.
- Todos los nodos internos menos la raíz y las hojas tienen a lo sumo ***m*** hijos (no vacíos) y como mínimo ***m***/2 hijos (no vacíos).
- El número de claves en cada nodo interno es uno menos que el número de sus hijos y estas claves dividen las de los hijos a manera de un árbol de búsqueda (1 clave $\Rightarrow$ 2 hijos, 2 claves $\Rightarrow$ 3 hijos).
- La raíz tiene como máximo ***m*** hijos. Si no es una hoja, tiene como mínimo 2 hijos y ninguno si el árbol consta de la raíz solamente.
- El orden de un árbol B es igual al máximo número de claves por nodo más uno.

La principal diferencia que tiene con los árboles AVL es en cuanto al manejo de datos dentro de los nodos; guardan las claves de determinados grupos de datos. La condición de que todas las hojas se encuentren en un mismo nivel impone un comportamiento en árboles B, a diferencia de los árboles de búsqueda, ya que no pueden crecer en sus hijos y se ven obligados a crecer por la raíz.

## Operaciones sobre un árbol B

Las operaciones básicas sobre un árbol B son las siguientes:

- Búsqueda de una clave
- Inserción de una clave nueva
- Eliminación de una clave

### *Búsqueda de una clave*

La búsqueda de una clave se realiza del mismo modo que en los árboles de búsqueda binarios. La única diferencia es que en cada nodo del árbol B existen varias claves, y habrá que determinar cuál de los punteros adyacentes a una clave en un nodo se utiliza para pasar al nodo hijo, en caso de que sea necesario. El proceso consiste en iniciar la búsqueda en el nodo raíz. Llamaremos Q a la clave que estamos buscando. Se recorre el nodo buscando la primera clave con valor mayor que Q. Si esta clave existe y es igual a Q finaliza el proceso. Si la clave existe, pero es distinta de Q, se repite el proceso en el nodo hijo apuntado por el puntero de la izquierda de la clave. Si la clave no existe, se continúa el proceso en el nodo hijo apuntado por el puntero de la derecha de la última clave del nodo. El proceso finalizará cuando se encuentre el valor buscado o se recorra completamente un nodo hoja.

### *Alta o Inserción de una clave nueva*

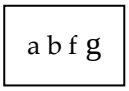
Primero se realiza una búsqueda para determinar con criterio de árbol de búsqueda en qué lugar se dará de alta la clave. Si la clave es nueva, la búsqueda del lugar va a terminar en una hoja del árbol, en ese momento

la clave se intenta añadir a la hoja del árbol a la cual se llegó. Si el nodo no está lleno, se da de alta. Si la clave debe ser agregada en un nodo lleno, éste se divide en dos nodos a un mismo nivel y la clave mediana no se coloca en ninguno de los dos nuevos nodos, sino que sube un nivel para insertarse en el nodo padre.
La inserción de una clave se realiza desde las hojas y en sentido ascendente en el árbol. Para insertar una nueva clave hay que localizar en primer lugar la posición en el árbol en el que la clave debería insertarse. Por tanto, lo primero que hay que hacer es una búsqueda en él árbol. En caso de que la clave no se encuentre en el árbol, se procede a su inserción.
Una vez determinado el punto de inserción en el nodo hoja correspondiente, se procede del siguiente modo:

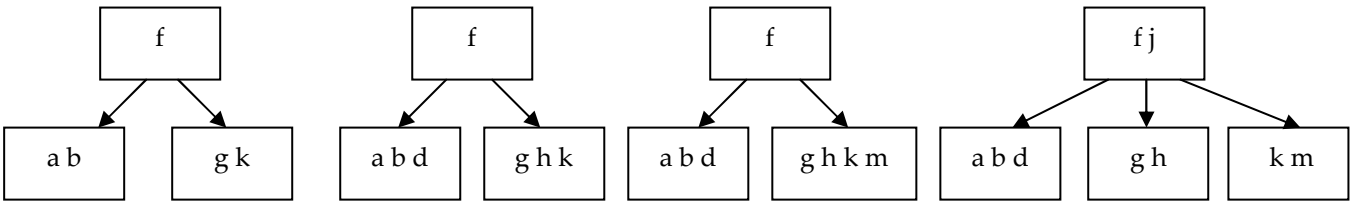
- Si el nodo hoja no está completo (es decir, si no tiene ***m***-1 claves para un nodo de orden ***m***), se inserta la clave en el lugar correspondiente.
- Si el nodo hoja está completo, entonces se inserta en el lugar que corresponde y se divide el nodo en dos nodos distintos con la mitad de las claves en cada uno de ellos, y se promociona la menor de las claves del nodo derecho resultante de la división al nodo padre.
- Si el nodo padre no está completo, la inserción ha finalizado. Sin embargo, si el nodo padre estuviese completo, habría que repetir el paso anterior hasta llegar a un nodo que no estuviese completo.
- En el peor de los casos, es posible que haya que promocionar claves hasta llegar al nodo raíz, en cuyo caso el nodo raíz se separaría en dos nodos distintos, y se promocionaría una clave a un nodo superior, que habría que crear y sería el nuevo nodo raíz.

*Ejemplo:*

Árbol B de orden 5. Secuencia <a, g, f, b, k, d, h, m, j>




Luego, viene ***k***, pero como el orden es 5, el nodo está lleno y ***f*** es la mediana, entonces el nodo se divide en dos y ***f*** sube al nodo padre **(a)**. Luego vienen ***d*** y ***h***, y se insertan en sus lugares respectivos sin que se supere en número máximo de claves por nodo **(b).** Lo mismo ocurre cuando se debe insertar *m* **(c).**
Viene ***j*** que debería insertarse entre ***h*** y ***k,*** pero como el orden es 5, el nodo está lleno y ***j*** es la mediana, se divide el nodo en dos y ***j*** sube al nodo padre (**d).**



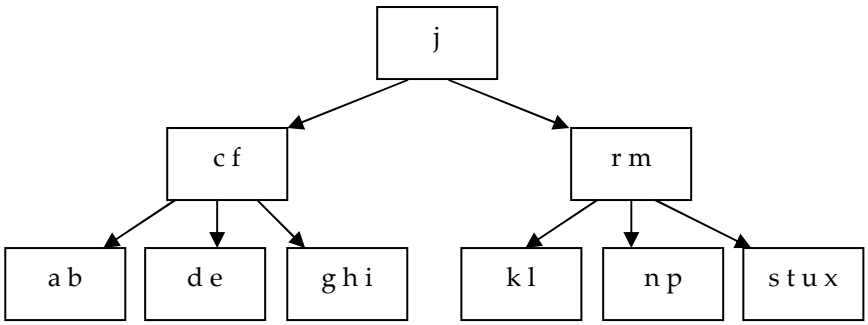

### *Baja o Eliminación de una clave*

Al igual que la inserción, la eliminación de una clave en un nodo puede resultar en situaciones no permitidas para un árbol B (por ejemplo, que el número de claves en el nodo en el que se realiza la eliminación quede con menos de ***m***/2 claves si el árbol es de orden m). Hay que considerar que siempre se debe mantener la forma o figura de árbol B, no importa qué elemento se quiere dar de baja, sino que los que queden cumplan con las condiciones de árbol B.
Primero se debe ubicar el elemento a dar de baja y posicionarse en ese nodo, luego se borra el elemento y se controla que el árbol siga siendo árbol B. Si al eliminar un elemento se pierden estas condiciones, es necesario reorganizarlo; se debe tratar que quede de la menor altura posible, siempre cuidando que no se viole la definición de árbol B.
Hay que seguir, por tanto, un procedimiento general para evitar que estos problemas se produzcan:

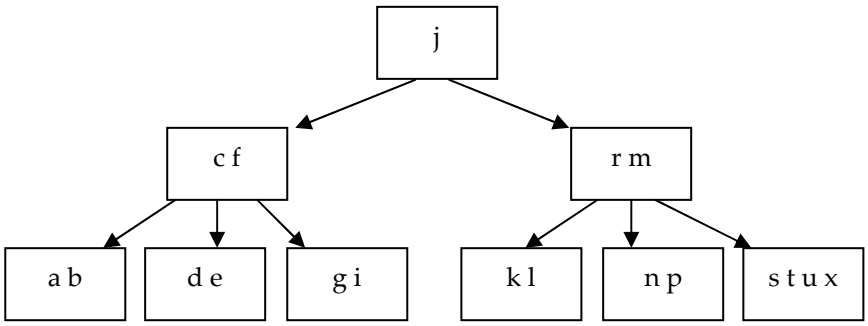
- Si el nodo en el que se va a realizar la eliminación contiene después de la operación de borrado al menos ***m***/2 hijos, (***m***/2 – 1 claves) el proceso termina.
- Si por el contrario no se produce la situación anterior, entonces es necesario reestructurar el árbol para que siga manteniendo su estructura de árbol B. Para ello se trabaja de dos modos: el primero trata de crear un único nodo tomando el nodo del que se elimina la clave y un nodo hermano, y luego se re-estructura el padre; el segundo crea dos nodos tomando el nodo del que se elimina la clave y un nodo hermano, y re-estructurando tanto los dos nuevos nodos como el nodo padre.

Algunos casos particulares son:

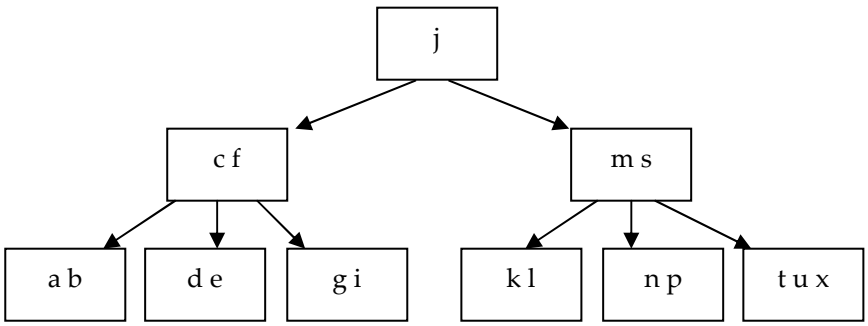
- Si la hoja contiene más del número mínimo de claves, una puede eliminarse sin acciones posteriores. Por ejemplo: Árbol B de Orden 5. Secuencia a eliminar <h, r, p, d>




Si se elimina la clave ***h***, el árbol no se modifica resultando la siguiente estructura:

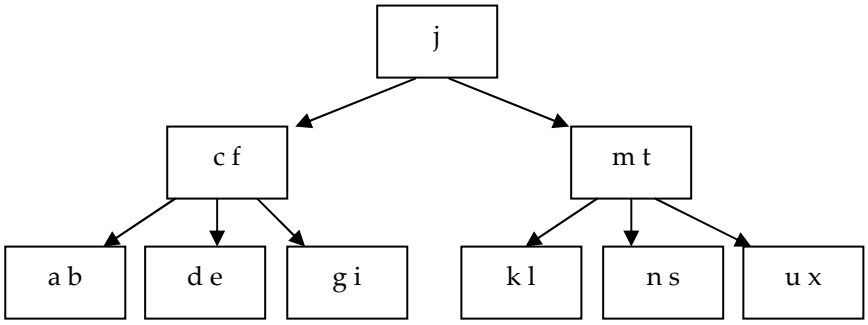



- Si la clave que se va a eliminar no está en una hoja, su predecesor o su sucesor (inmediato) será una hoja, por lo tanto se elimina la clave y se pone el predecesor en su lugar. Por ejemplo, si se elimina la clave ***r***, sube ***s***, se reacomodan las demás claves afectadas en la operación y resulta:

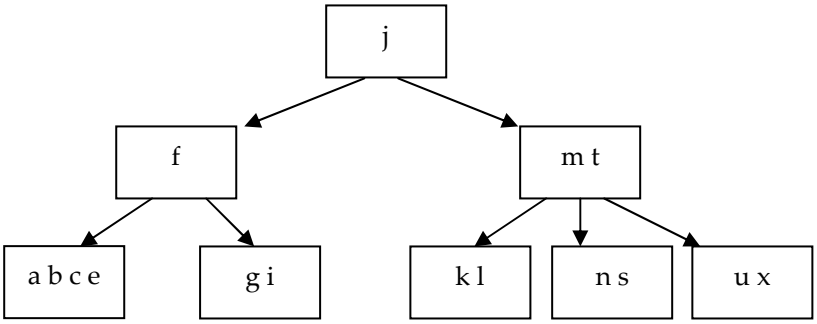



- Si la clave que se va a eliminar está en una hoja que contiene el número mínimo de claves, se observan las dos hojas que son adyacentes. Si una excede el número mínimo de claves se puede poner una clave en

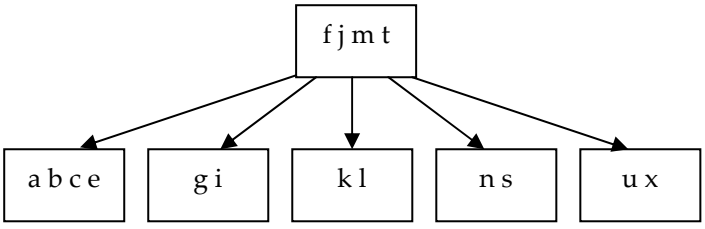
el nodo padre y la clave del padre en la hoja donde se realiza la eliminación. Por ejemplo, si se elimina la clave ***p***, sube ***t*** (clave de la hoja adyacente), y baja ***s*** (clave del nodo padre) en el lugar de ***p***.




- Si la clave que se va a eliminar está en una hoja que contiene el número mínimo de claves y además la hoja contigua tiene el numero mínimo de claves, las dos hojas y la clave mediana del padre se pueden combinar como una nueva hoja, entonces contendrá el numero máximo de claves permitidas. Por ejemplo, se debe eliminar ***d***, dicho nodo queda con una sola clave, entonces se combina este nodo con su adyacente izquierdo, y baja una clave del nodo padre ***c***.



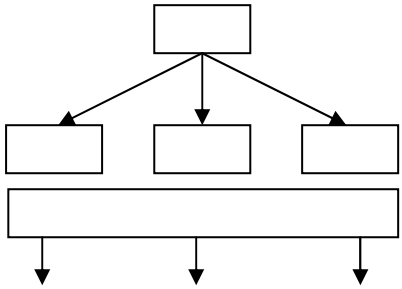

- Si esto deja al nodo padre con pocas claves (desproporcionado), el proceso se propaga hacia arriba, en un caso límite disminuye la altura del árbol (como son árboles de búsqueda, cuando menor sea la altura, menos costosa serán las operaciones sobre árbol). Así, disminuye en un nivel al árbol, ya que al reacomodar desaparece el nivel I.




**Nota:** no puede haber un nodo con un solo elemento salvo la raíz (de alguna forma se puede reorganizar).

## USO DEL ARBOL B EN ARCHIVOS

Para convertir un árbol B en índice hay que tener en cuenta que los nodos terminales pueden asociar direcciones de espacio de almacenamiento de disco.
En el último nivel se forman las claves completas. A partir de la raíz, la suma de las claves internas hasta un nodo terminal puede ser la clave más completa.



**Dirección de Disco**

En la realidad, el área de índices (AI) en un árbol B, es un árbol de búsqueda en el cual sus ramas contienen información de cómo se van generando los índices.
Los nodos terminales tienen asociadas las direcciones físicas de los registros (la dirección física del área de datos), asociadas a las claves.

**¿Por qué se utiliza un árbol y no una tabla?**
Esto sucede debido a que la estructura de árbol es más rápida que la estructura de una simple tabla.
Al sistema operativo no le interesa que el espacio para almacenar los registros (AD) sea contiguo. Son bloques de memoria direccionados al área de índices.
Si se pierde el encadenamiento, quedan bloques dispersos en memoria; el administrador de archivos se encarga de mantener, por medio de un campo, el encadenamiento entre los bloques pudiendo reconstruir el AI.

# TEMA V: ESTRUCTURAS IRRESTRICTAS

Son grafos en los que no es restricción la forma en que los nodos están conectados. Para este tipo de estructura no hay un nodo de inicio condicionado; generalmente, el primer nodo dado de alta es el nodo de acceso.

***Diseño de Celdas de Pfaltz:*** se basó en la idea de que si se tiene una estructura de la que no se sabe su grado de entrada y su grado de salida, por lo tanto no puede existir un único diseño de celdas. Entonces la idea es tener dos diseños de celda, uno para los nodos y otro para los arcos.
Consiste en una celda para la información (Celda de nodo) y otra para el enlace (Celda de arco). El diseño de celda de *Pfaltz* surge debido a que a un nodo le puede llegar ***x*** cantidad de arcos y de él puede partir ***n*** cantidad de arcos, lo cual en una celda simple tiene un costo alto de implementación y no es útil ni eficiente.

## Celda de Nodo

| | |
|---|---|
| **Ledge** | **Redge** |
| **Ident_Nodo** | |
| **$f_i$** | |
| **Tipo** | **Next_N** |

***Ident_Nodo (Identificador de Nodo):*** nombre del nodo.
***$f_i$:*** funciones de asignación a nodo. Se divide en tantas partes como funciones tenga.
***Tipo:*** identificador que permite determinar el tipo de celda (nodo o arco).
***Next_N:*** es un encadenamiento entre celdas de nodo a medida que éstas se van dando de alta (sin importar la estructura del grafo). Otra definición: es un puntero que permite encadenar la celda con el resto de celdas de nodo en un determinado orden y sin repeticiones.
***Ledge*** *(conjunto de llegada):* es un puntero a una lista cuyas celdas contienen las direcciones de las celdas de arcos que llegan a la celda de nodo considerado.
***Redge*** *(conjunto de salida):* es un puntero a una lista cuyas celdas contienen las direcciones de las celdas de arcos que parten del nodo considerado.

## Celda de Arco

| | |
|---|---|
| **LPoint** | **RPoint** |
| **LLink** | **RLink** |
| **Ident_Arco** | |
| **$g_i$** | |
| **Tipo** | **Next_A** |

***Ident_Arco:*** nombre del arco (representa un identificador único para el arco).
***$g_i$:*** funciones de asignación a arco. Dividido en tantas partes como características tenga.
***Tipo:*** identificador que permite determinar el tipo de celda (nodo o arco).
***Next_A:*** es un encadenamiento entre celdas de arco a medida que se van dando de alta. Otra definición: es un puntero que encadena la celda con el resto de las celdas de arco manteniendo un orden y formando una estructura.
***LPoint:*** puntero a una celda de nodo del cual parte el arco considerado.
***RPoint:*** puntero a una celda de nodo al cual llega el arco considerado.

***LLink:*** puntero a una lista cuyo contenido son las direcciones de las celdas de arcos que llegan al mismo nodo que el arco considerado.
***RLink:*** puntero a una lista cuyo contenido son las direcciones de las celdas de arcos que salen del mismo nodo que el arco considerado.

***Características***

- De un nodo, pueden salir ***m*** cantidad de arcos y a un nodo pueden llegar ***n*** cantidad de arcos.
- Pueden existir ciclos y bucles en la estructura.
- Recorrido: no se puede pasar directamente de un nodo a otro. Se debe pasar por las listas que encadenan las salidas, hasta encontrar la celda que llega al otro nodo, es decir, que se debe pasar por las celdas de arco. Para ver que arcos llegan al nodo, se debe recorrer ***Ledge***.

***Dinámica:***

*Forma de acceso:* es el primer tipo de estructura que no se rige por un referente o lugar de acceso; esto no quiere decir que tenga varios punteros de acceso pero es más flexible porque se puede elegir el nodo de acceso para luego desplazarse en la estructura.
**Hay dos formas de llegar desde un nodo hasta otro:**

- *Por medio de las relaciones:* es decir, se accede a la celda nodo y a través de ella a la celda de arco, luego a la celda nodo, luego a la celda arco, y así hasta llegar al nodo buscado. (Desventaja: puede suceder que no sea posible llegar, por ejemplo si el nodo es un minimal).
- *Por medio del campo Next_N:* ésta es la forma más eficiente, ya que como no toma en cuenta la estructura del grafo, es más rápida. Además, es posible acceder a todos los nodos existentes en el grafo.

***Altas***: para dar de alta a una celda de nodo es necesario reservar memoria de acuerdo a un tamaño determinado y a partir de ahí grabar la información (dato). Se pueden dar primero de alta todos los nodos y posteriormente las relaciones, o se pueden dar da de alta simultáneamente los nodos y relaciones (resulta más ordenado dar de alta todos los nodos primero).

- **Alta de nodo**
  - Crear la celda;
  - Cargar la información;
  - Actualizar el grupo Next.
- **Alta de arco**
  - Verificar la existencia origen y destino;
  - Crear la celda;
  - Actualizar LPOINT y RPOINT;
  - Actualizar el Ledge del destino: tomar el Ledge del destino, copiarlo generando la nueva lista que conforma el RLink del nuevo arco. Agregar el nuevo arco del Ledge del destino; visitar cada uno de los arcos contenidos en el Ledge, actualizando su RLink con el nuevo arco.
  - Actualizar el Redge del origen: secuencia similar a la anterior;
  - Actualizar campo Next.

**Bajas:**

- **Baja de nodo**
  - Localizar el nodo a dar de baja.
  - Tomar el Ledge;
  - Eliminar cada uno de los arcos a los cuales se apunta en la lista;
  - Eliminar la lista;
  - Tomar el Redge; ídem Ledge;
  - Actualizar la lista de nodos;
  - Eliminar el nodo.
- **Baja de arco**
  - Localizar el arco a dar de baja;

- Con el LPOINT se accede al nodo origen y se actualiza su REDGE;
- Con el RPOINT se accede al nodo destino y se actualiza su LEDGE;
- Se toma el LLINK y se accede a cada uno de los arcos de los que allí se referencia, actualizando sus LLINK;
- Eliminar el LLINK;
- Tomar el RLINK y acceder a cada uno de los arcos de los que allí se referencia, actualizando sus RLINK;
- Eliminar el RLINK;
- Actualizar el Next_Arco;
- Dar de baja el arco.

## BUSQUEDA DE PASO EN ESTRUCTURAS IRRESTRICTAS

*Búsqueda en Profundidad –DEPTH FIRST–:* determina si existe "paso de $x$ a $y$" y lo lista. Toma una de las ramas (caminos – zonas) y explora el grafo hasta sus nodos finales. Si no encuentra un paso de $x$ a $y$, tiene la capacidad de retroceder para buscar en otra rama (no cae en ciclo). Se caracteriza por su rapidez y el poco uso de memoria.

**Pseudocódigo:**

```
s ← ^                          // s: pila; closed: lista; exito: variable
exito ← falso
closed ← x
s ← x
while s ≠ ^ or Not exito do
    p ← s
    if p = y then
        s ← p
        exito ← verdad
        else
            if R (p) ≠ ∅ then
                s ← p
                n ← R (p)
                R (p) ← R (p) + {n}
                if n ∉ closed then
                    closed ← n
                    s ← n
                End if
            End if
    End if
End while do
End Pseudocódigo.
```

*Búsqueda a lo ancho –BREADTH FIRST–:* hace un barrido a lo ancho del grafo (por niveles) examinando todos los pasos posibles entre el nodo inicial y el nodo buscado. A medida que se va avanzando, se almacenan todas las condiciones de paso de un nodo a otro. Es más lenta, ocupa más memoria y solamente determina si existe paso, no lo lista.

**Pseudocódigo:**

```
open ← x
exito ← falso
```

```
while open ≠ ^ or exito do
    p ← open
    closed ← p
    if p = y then
        exito ← verdad
        else
            M ← R (p)
            M ← M – {open} U {closed}
            open ← open + M
    End if
End while do
End Pseudocódigo
```

*Búsqueda con heurística:* requiere que la estructura tenga información adicional para saber como recorrer. Es sumamente buena y simplifica algoritmos pero tiene la desventaja de requerir del aporte humano como soporte de la búsqueda. Esto implica que depende del conocimiento que se pueda aportar.
Heurística = tarea de valorizar - ponderar - aporte de conocimiento.
Por ejemplo: si además de tener los nodos se tiene asignado un determinado valor para los arcos, es posible facilitar el acceso por el camino más adecuado.

**Funciones que debe cumplir un algoritmo que retorne el paso entre 2 nodos cualesquiera:**

**a.** Verificar que existe un nodo ***x*** (origen) y un nodo ***y*** (destino).

**b.** Tomo el nodo ***x*** y leo el ***Redge*** (dirección de una lista), que tiene todas las direcciones de arco que salen de él:
- Si no hay ningún elemento, no hay paso.
- Si hay elementos, tomo el primero, voy al nodo que comunica el origen con la dirección de arco que leí, si ese nodo es igual a ***y*** encontré un paso, si no leo su ***Redge***, si está vacía la lista de direcciones de arco y el último nodo leído no es ***y***, no hay paso, vuelvo hacia atrás a leer otra dirección del nodo ***x***, y así sucesivamente hasta que la lista de direcciones de arco ***x*** sea ***Nil***, o encuentre el paso.

**c.** Cada vez que encuentro un arco que sale de ***x*** y el nodo donde llega tiene salida, lo guardo en una lista auxiliar (siempre y cuando no sea ***y***) para luego informar la secuencia de paso.

El algoritmo **Sort Topológico,** o de **Clasificación Topológica,** es un proceso de asignación de un orden lineal a los nodos de un grafo dirigido acíclico, tal que si existe un arco del nodo i al nodo j, i aparece antes que j en el ordenamiento lineal (i < j).
La técnica del Sort Topológico consiste en ir tomando los mínimos e informarlos, luego se destruye el mínimo, se busca el siguiente, se informa y se destruye y así sucesivamente. Esto da como resultado el ordenamiento topológico destructivo.
Existe la misma técnica no destructiva, que consiste en ir marcando el nodo que ya se informó, para no volverlo a tomar.
Cuando no hay mínimos, sino que existen minimales, se toma cualquiera del conjunto minimal (no hay condiciones para elegirlo) para iniciar la secuencia, luego se informa y se destruye o se marca.
Dos ordenamientos topológicos del mismo grafo pueden dar resultados diferentes (distintas secuencias), en el único caso es en el de un grafo finito conectado. El Sort termina cuando se visitaron todos los nodos.
El **algoritmo Warshall** se aplica en grafos acíclicos (libres de loops). Se orienta a recuperar información o acceder a la misma. Es la adaptación de trabajo con matrices cuadradas, que determina las posibles longitudes de paso y, además, determina la existencia de un paso entre dos nodos. No devuelve la secuencia, devuelve la existencia o no del paso entre dos nodos.
Se parte de una matriz A que representa las relaciones del grafo. Multiplicando esta matriz por sí misma se representan las distintas longitudes de paso (A2 pasos de longitud 2, A3 pasos de longitud 3). Los elementos de la matriz indican la cantidad de pasos de un nodo a otro. El mérito de Warshall es la fórmula o algoritmo, que es una forma más práctica, fácil y rápida para el producto de matrices.

# TEMA VI: NORMALIZACIÓN

***Introducción:*** la normalización de estructuras de datos es una técnica determinística que simplifica estructuras de datos complejas, dividiéndola en dos o más estructuras simples. Ejemplo: un acta de examen, factura, remito, etc.
**Identificador o clave primaria:** es un conjunto de campos que identifican unívocamente a cada registro del archivo.
**Propiedad de unicidad:** toda combinación de valores de la clave primaria debe referenciar a un único registro del archivo. Ejemplo: en un archivo de personas, el DNI (no, por ejemplo, el sexo).
**Propiedad de minimalidad:** esta propiedad dice que si la clave primaria es compuesta, debo probar suprimir uno a uno los elementos de la clave y si en todos los casos pierdo la condición de unicidad, esta clave es mínima.

La normalización es una técnica que se ha desarrollado para obtener estructuras de datos eficientes. El concepto de normalización fue introducido por E. F. Codd, y fue pensado para aplicarse a sistemas relacionales. Sin embargo, tiene aplicaciones más amplias. La normalización es la expresión formal del modo de realizar un buen diseño. Provee los medios necesarios para describir la estructura lógica de los datos en un sistema de información. Las ventajas de la normalización son:

- Evitamos dependencia de inserciones, actualizaciones y borrados.
- Se reduce la reestructuración en la base, debida a la introducción de nuevos datos. Se mejora la dependencia de los datos, permitiendo realizar extensiones de la base de datos, afectando muy poco o nada a los programas d aplicación existentes que acceden a la base de datos.
- No se establecen restricciones artificiales en la estructura de los datos.

Los principios básicos de la normalización se utilizan ampliamente como instrumento para el diseño de sistemas de información. Aunque la técnica se ideó para sistemas relacionales, debe recalcarse que puede utilizarse como instrumento de diseño para cualquier sistema de información. Se puede utilizar con ficheros tradicionales, al igual que con sistemas de gestión de bases de datos.
La normalización involucra tres fases que se realizan en orden. La realización de la segunda fase supone que la primera fase ya se ha completado, y lo mismo le ocurre a la tercera con respecto a la segunda. Tras completar cada fase, se dice que los datos se encuentran en primera, segunda o tercera forma normal. Éstas se abrevian por 1FN, 2FN, 3FN o por PFN, SFN, TFN (en inglés, FNF, SNF, TNF). El documento original sobre normalización de E. F. Codd, proponía las tres fases. Se ha definido una Cuarta Forma Normal, pero no ha tenido una aceptación general y no la consideraremos.
El proceso de normalización de Codd consiste en tres etapas o reglas que transforman unas relaciones no normalizadas en otras expresadas en Tercer Forma Normal (TFN).

## PRIMERA FORMA NORMAL

La esencia de la Primera Forma Normal es que:
"Un registro en la Primera Forma Normal no incluye ningún grupo repetitivo".
***Definición formal:*** ***"Una relación está en Primera Forma Normal, si cumple la propiedad de que sus dominios no tienen elementos que, a su vez, sean conjuntos".***
Una relación que no esté en Primera Forma Normal se dice que no está normalizada. Una relación normalizada se puede presentar como un fichero plano tal que cada n-tupla es un registro con un formato fijo.
La eliminación de los grupos repetitivos hace que las relaciones sean más fáciles de comprender, ya que los datos se pueden expresar como un conjunto de tablas simples.
Físicamente, las relaciones en PFN se pueden instrumentar como un fichero plano con registros de longitud fija. La forma no normalizada se podría representar por un tipo fichero de registros múltiples o por un fichero con registros de longitud variable. Ambos métodos incrementan la complejidad de los mecanismos de almacenamiento y de los programas que acceden a la base. Los registros de longitud variable tienen, generalmente, un límite superior respecto a la longitud.

La PFN presenta problemas con las tres operaciones básicas de almacenamiento: inserción, borrado y almacenamiento.

## SEGUNDA FORMA NORMAL

La esencia de la primera forma normal es que:
En cualquier registro que esté en Segunda Forma Normal (SFN), todos sus campos (atributos o elementos de datos) dependen de la clave completa y no sólo de una parte de ésta.
***Definición formal:*** ***"Se dice que una relación R está en Segunda Forma Normal, si además de estar en la Primera Forma Normal, cualquiera de sus atributos no-primarios tienen una dependencia funcional plena con cada una de las claves candidatas de R".***
Un atributo no-primario es aquel que no forma parte de la clave. Esta definición nos dice que todo elemento de dato (atributo) debe depender de la clave completa, y no sólo de una parte. Se deben eliminar las dependencias de claves parciales. Este paso se aplica sólo a relaciones con claves compuestas. Una relación que esté en PFN y que no tenga claves compuestas ya está en SFN.
Los problemas planteados con la PFN se resuelven con la SFN, pero todavía se presentan problemas con la SFN con la inserción, borrado y actualización.

## TERCERA FORMA NORMAL

La Tercera Forma Normal es una extensión de la Segunda Forma Normal. La Segunda Forma Normal elimina las dependencias respecto a una clave parcial. La Tercera Forma Normal (TFN) elimina las dependencias entre atributos no primarios (no pertenecientes a la clave).
Para todo campo de una relación que esté en Tercera Forma Normal, se cumple que depende sólo de la clave y de ningún otro atributo de la relación. Un registro que esté en Tercera Forma Normal, también está en SFN.
***Definición formal:*** ***"Una relación R está en Tercera Forma Normal si está en Segunda Forma Normal y además ninguno de sus campos no-primarios tiene dependencias transitivas respecto de las claves candidatas de R".***
Un atributo o campo no-primario es aquel que no pertenece a la clave. Si A→B y B→C, implica que A→C, entonces se dice que C depende transitivamente de A. En otras palabreas, si C depende de B, y B depende de A, implica que C depende de A, entonces decimos que C depende transitivamente de A.
La definición dice que no debe haber dependencias transitivas entre elementos de datos que no forman parte de la clave (atributos no-primarios). La SFN se ocupa de las dependencias de los elementos de datos con la clave. Sin embargo, la TFN se ocupa de las dependencias entre elementos de datos no-primarios de la relación.

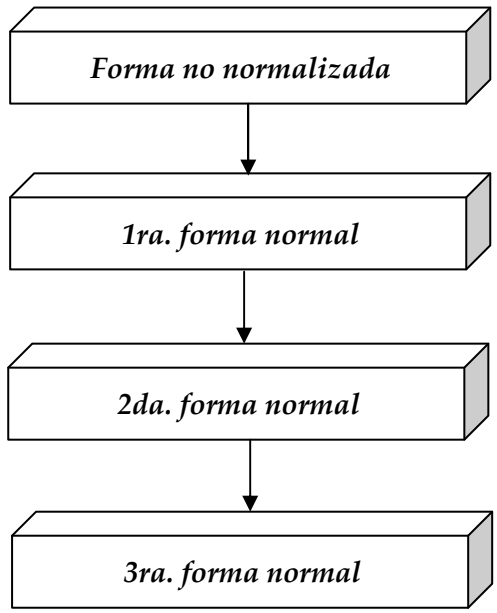
## NORMALIZACIÓN PRÁCTICA

La normalización es un instrumento práctico que nos permite organizar y establecer una estructura de datos y un modelo de datos. Aunque se desarrolló para utilizarse en sistemas relacionales, esta metodología puede aplicarse fuera de este entorno.
La normalización es una base para muchas de las técnicas de diseño de sistemas formales que actualmente se encuentran en el mercado.
Un método práctico típico utiliza la normalización en el análisis de los datos. En primer lugar, hay que buscar las relaciones naturales existentes entre los datos, olvidándonos de consideraciones de diseño futuras. En otras palabras, se quieren obtener las relaciones puras y naturales. Éstas se expresan mediante relaciones no normalizadas.
Las relaciones no normalizadas se transforman en Primera Forma Normal (PFN) creando;
1. Una relación para los campos que sean únicos.
2. Una relación para cada campo repetitivo.

Si la relación contiene una clave compuesta, se debe transformar en la Segunda Forma Normal.
Esto se hace creando una tabla para todos aquellos atributos que dependan completamente de la clave compuesta. El proceso se repite para aquellos atributos que dependen de parte de la clave compuesta.

Ahora ya tenemos un conjunto de relaciones en Segunda Forma Normal (SFN). Entonces se examinan todas las tablas para ver si hay campos no primarios que dependan unos de otros. Si se encuentran, se crea una nueva tabla para ellos. Así obtendremos relaciones en TFN.
Ya en esta etapa se puede optimizar la TFN. Puede haber relaciones degeneradas que contengan sólo la clave; generalmente se pueden eliminar. Puede que otras relaciones tengan la misma clave, por lo que se pueden combinar en una sola, siempre que el resultado sea lógico y tenga sentido. Si en este proceso de optimización se descubren anomalías, generalmente son debidas a ambigüedades subyacentes en los datos.
Los analistas y diseñadores con experiencia producirán relaciones en TFN, sin saber que lo están haciendo; y es que utilizan el sentido común y la experiencia para escribir relaciones normalizadas. La técnica de normalización formal proporciona medios para verificar la solidez de tales soluciones.
El modelo de datos, generado mediante el proceso de normalización, proporciona medios para introducir cambios surgidos al crearse nuevas demandas o cambiar las circunstancias. Se pueden realizar modificaciones en el modelo y comprobar si todavía sigue estando en TFN. Las modificaciones pueden involucrar adiciones en las relaciones ya existentes, o la creación de nuevas tablas, y probablemente ambas cosas.
La normalización es una técnica útil en cualquier circunstancia. Es casi esencial si se quieren aprovechar al máximo las ventajas que ofrece el diseño relacional.

## PASOS EN EL PROCESO DE NORMALIZACIÓN




1. Reducir todas las estructuras de datos que no sean bidimensionales a relaciones o segmentos bidimensionales.

2. Eliminar cualquier dependencia incompleta de atributos de claves candidatas.

3. Eliminar cualquier dependencia transitiva de atributos no primarios de clases candidatas.

# ANEXO

## FORMULAS Y DEMOSTRACIONES IMPORTANTES

La cantidad máxima de flechas que puede haber en un grafo G finito con una única relación = $n^2$. Para excluir los loops se debe restar n, entonces $n^2 - n$ es la cantidad máxima de flechas que puede haber en un grafo G finito con una única relación sin incluir los loops.
Nota: n es la cantidad de nodos o puntos del modelo

1. **Demostrar que $\exists\rho(x,z) \Leftrightarrow \overline{R}(x) \cap \overline{L}(z) \neq \phi$ a partir de las definiciones de la Teoría de Grafos.**
   No se puede demostrar una afirmación que no es válida para todos los casos:
   Por ejemplo, si tenemos un grafo G = (P, R) donde P = {$x$, $z$} y R = {($x$, $z$)} entonces
   $\exists\rho(x,z)$; $\overline{L}(z) = x$ y $\overline{R}(x) = z$ $\therefore \overline{L}(z) \cap \overline{R}(x) = \phi$ con lo cual concluimos que no verifica la afirmación planteada.

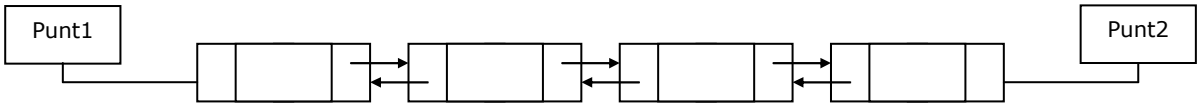
2. **¿Cuál es la longitud máxima de un paso en un grafo finito? Justificar y demostrar.**
   La longitud máxima de un paso en un grafo finito es ∞ esto se debe a la presencia de ciclos.
   **Demostración:**
   La existencia de ciclo ⇒ que ∃ x, y ∈ P / ∃ ρ (x, y) ∧ ∃ ρ (y, x) ∴la secuencia
   **(1)** $x, z_1, z_2, \ldots, z_n, y, w_1, w_2, \ldots, w_m, x$
   **(2)**
   define un ciclo de longitud $(n+1)+(m+1)$, a su vez
   $x, z_1, z_2, \ldots, z_n, y, w_1, w_2, \ldots, w_m, x, z_1, z_2, \ldots, z_n, y, w_1, w_2, \ldots, w_m, x$
   define un ciclo de longitud $2(n+1)+2(m+1)$ y así sucesivamente al ciclo anterior se le puede agregar K veces la cadena **(1)**, siendo K un número $Z^+$. Por lo cual la longitud de la secuencia estará definida por $K(n+1)+K(m+1)$ y como K pertenece a los $Z^+$ la longitud de paso puede tomar cualquier valor incluso ∞.

3. **Sea el grafo G = (P, R), G es lineal ⇔ G es básico.** Esta afirmación es falsa debido a que un grafo básico puede tener más de una entrada a un nodo. Por ejemplo si tenemos un grafo G = (P, R) donde P = {$a$, $b$, $c$}y R = {($a$, $c$), ($b$, $c$)} entonces G es básico pero no lineal ya que |L(c)| = 2, con lo cual se viola una de las condiciones de Grafo Lineal.

4. Una estructura irrestricta (por ejemplo, los diseños de celda definidos por Phaltz) puede utilizarse para implementar un árbol binario ya que toda estructura irrestricta es un grafo en el que no es restricción la forma en que los nodos están conectados.
5. Integridad: característica de las bases de datos que indica la ausencia de datos inconsistentes.

6. Existen datos que se utilizan en distintos sitios, la **consistencia** implica que esos datos son iguales en todos los lugares de trabajo, al mismo tiempo.

7. Una cola de doble entrada no es una estructura lineal, ya que sus elementos tienen más de un antecesor y más de un sucesor.



8. Es posible que el grado de entrada de un nodo $x$ de un árbol sea ≥ 2, si estamos hablando de árbol principal izquierdo.

**9. Diferencia (funcionales y estructurales) entre AVL y Árbol B**

| AVL | Árbol B |
|---|---|
| Orden 2 (binario) | orden m |
| cada nodo almacena una única clave | cada nodo puede almacenar como máximo $(m-1)$ claves |

La principal diferencia que tiene un Árbol B con un AVL es en cuanto al manejo de datos dentro de los nodos.
La condición de que todas las hojas se encuentren en un mismo nivel impone un comportamiento en árboles B, a diferencia de los árboles de búsqueda, ya que no pueden crecer en sus hijos y se ven obligados a crecer por la raíz.

**10. Demostrar que $\exists\rho(x,z) \Leftrightarrow \overline{R}(x) \cap \overline{L}(z) \neq \phi$ sabiendo que P = {x, y, z} a partir de las definiciones de la Teoría de Grafos.**

**1º Parte:** si $\exists\rho(x,z) \rightarrow \overline{R}(x) \cap \overline{L}(z) \neq \phi$

Aseveramos que $\exists\rho(x,z) = \{x_0, x_1, ...x_n\}$ donde $x = x_0$ y $z = x_n$ entonces podemos determinar que

$$\overline{R}(x) = \{^y / \exists\rho(x,y)\} = \{x_1, x_2, ....., x_n\} \text{ y que}$$

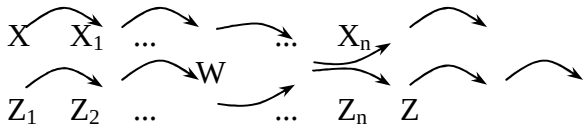
$$\overline{L}(z) = \{^y / \exists\rho(y,z)\} = \{x_{n-1}, ....., x_1, x_0\} \text{ y por lo tanto que}$$

$$\overline{R}(x) \cap \overline{L}(z) = \{x_1, x_2, ...., x_{n-1}\} \neq \phi$$

Concluimos entonces que esta primera parte es cierta pero se debe demostrar el otro camino.

**2º Parte:** si $\overline{R}(x) \cap \overline{L}(z) \neq \phi \rightarrow \exists\rho(x,z)$

Como $\overline{R}(x) \cap \overline{L}(z) \neq \phi$ entonces $\exists w / w \in \overline{R}(x) \wedge w \in \overline{L}(z)$ y por lo tanto al tener un punto en común existe una secuencia de puntos que permiten ir de $x$ a $z$, es decir existe el conjunto $\{x, x_1, x_2, x_3, ..., w, ...z_n, z\}$ que define un paso de $x$ a $z$.




**11. Sea el grafo G = (P, R), G es árbol ⇒ G es básico**

Si un grafo G es árbol entonces cuenta, entre otras, con las siguientes dos propiedades, cada arco es un arco desconectante, esto implica que todo árbol está ***libre de loops***, y dados dos puntos cualesquiera $x$, $z \in$ a P, existe camino de $x$ a $z$ y ese camino es único, esto nos está indicando que ***no existen arcos redundantes***. Para concluir las dos condiciones que deben cumplirse para que un grafo sea básico son éstas, por lo tanto la afirmación es correcta.

**¿Una multilista es un hipergrafo?**

Considerando la definición de hipergrafo, la cual dice que para un mismo conjunto de puntos pueden existir varias relaciones y a su vez funciones de asignación a nodos y funciones de asignación a arcos, y la definición de multilista, que nos dice que es una lista que contiene varias listas en su interior; podemos decir que una multilista es un hipergrafo.

**¿Las listas doblemente enlazadas son multilistas?**
No, ya que las listas doblemente enlazadas contienen dos campos de enlace y se puede avanzar y retroceder en la lista, precisamente porque contiene *links* al antecesor y al sucesor, cosa que no ocurre con las multilistas, que contienen listas como información y no se puede retroceder sobre la lista.

**¿Una lista doblemente enlazada es una estructura lineal?**
Una lista doblemente enlazada no es una estructura lineal, ya que sus elementos tienen más de un antecesor y más de un sucesor.