TEMA 12

ORGANIZACIÓN LÓGICA DE LOS DATOS. ESTRUCTURAS DINÁMICAS.

ÍNDICE

1. INTRODUCCIÓN

Se denomina **dato** a cualquier objeto manipulable por el ordenador. Un dato puede ser un carácter leído de un teclado, información almacenada en un disco, un número que se encuentra en memoria principal, etc.

Los tipos de datos más frecuentes utilizados en los diferentes lenguajes de programación son los **datos simples** (lógico, carácter, entero, real) y los **datos estructurados o compuestos**. Los tipos de datos simples o primitivos no están compuestos de otras estructuras de datos; son los más frecuentes y utilizados por casi todos los lenguajes. Los tipos de datos compuestos se construyen a partir de otros tipos de datos; un ejemplo es la cadena de caracteres.

Una **estructura de datos** es una colección de datos organizada de un modo particular y sobre la que se definen ciertas *operaciones*. Será **homogénea** cuando todos los datos elementales que la forman son del mismo tipo o **heterogénea**, en caso contrario.

Las estructuras de datos pueden ser de dos tipos:

- **Estáticas**

  Son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse durante la ejecución del programa. Estas estructuras están implementadas en casi todos los lenguajes: **registros**, **ficheros** y **tablas** (o **arrays**).

- **Dinámicas**

  Son aquellas cuya ocupación en memoria puede aumentar o disminuir en tiempo de ejecución.

  No tienen las limitaciones o restricciones en el tamaño de memoria ocupada que son propias de las estructuras estáticas. Mediante el uso de un tipo de datos específico, denominado **puntero**, es posible construir estructuras de datos dinámicas. Las estructuras de datos dinámicas por excelencia son las **listas** (enlazadas, pilas, colas), **árboles** (binarios, B) y **grafos**.

La elección del tipo de estructura de datos idónea a cada aplicación dependerá esencialmente del tipo de aplicación y en menor medida del lenguaje, ya que si no tiene implementada una estructura, deberá ser simulada con el algoritmo adecuado.

2. LISTAS

Una **lista** *está formada por un número variable de datos* (**elementos**) *de un mismo tipo, ordenados según una secuencia lineal*. Cada elemento, salvo el primero, tiene un predecesor en la lista. Todos los elementos, salvo el último, tienen un sucesor. La lista es una estructura dinámica. Se pueden expresar mediante una lista: los componentes de una máquina, el itinerario de un autobús, las letras de un abecedario, etc.

Formalmente podemos definir una **lista** como una *estructura de datos formada por registros de al menos dos campos, en los que uno de ellos contiene información que permite localizar al siguiente registro en la lista según una secuencia dada* (lista enlazada).

Con una lista se pueden realizar las siguientes **operaciones**:

- **Añadir un elemento**: Esto puede realizarse al final de la lista, al principio, o entre dos elementos (**inserción**).
- **Eliminar un elemento**: Al eliminar un elemento no se pierde la secuencia lógica. Esto es, el predecesor del elemento eliminado pasa a ser el predecesor del siguiente elemento al eliminado.
- **Acceder al primer elemento de la lista**: El primer elemento es normalmente el único al que se puede acceder directamente.
- **Acceder al elemento siguiente del último procesado**: Este es el mecanismo normal de acceso a la lista. Al acceder a un elemento, éste no se elimina. La lista se puede leer desde el comienzo tantas veces como sea necesario.
- **Saber si la lista está vacía**: La lista está vacía si no contiene ningún elemento.

Aunque a primera vista pueda parecer que una lista es semejante a un array lineal, éste es una estructura completamente diferente. Por un lado, la lista es una estructura dinámica, ocupa en memoria el espacio necesario para albergar los elementos que se le han añadido. Por otro lado, la lista no es

direccionable, tan solo se puede recuperar un elemento accediendo antes a los que le anteceden y, por tanto, en cada momento hay un solo elemento en disposición de ser procesado.

Se denomina **contigua** a una lista cuyos elementos se almacenan en posiciones consecutivas de la memoria. En este caso se procesa como un array unidimensional y tanto el acceso a cualquier elemento como la adición de elementos es una tarea fácil. Sin embargo, la inserción o borrado requerirá un desplazamiento de los elementos que le siguen y, en consecuencia, el diseño de algoritmos específicos.

2.1. Listas enlazadas

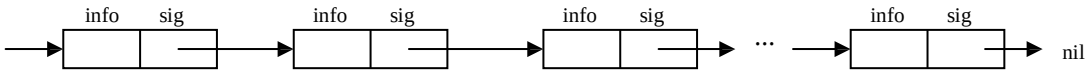
Los inconvenientes de las listas contiguas se eliminan con las listas enlazadas. Se pueden almacenar los elementos de una lista lineal en posiciones de memoria que no sean contiguas.

Una **lista enlazada** o encadenada *está formada por un conjunto de elementos en el que cada elemento contiene la posición del siguiente elemento de la lista.* Cada elemento tendrá al menos dos campos: uno con el valor del elemento y otro con la posición del siguiente elemento (*campo enlace*).

Con las listas enlazadas se utiliza una terminología propia usualmente. Los valores se almacenan en un **nodo**. Los componentes de un nodo se denominan **campos**. Un nodo tiene al menos un campo **dato** y un **enlace** (puntero) con el siguiente campo. El campo enlace apunta al siguiente nodo de la lista. El último nodo de una lista enlazada, por convenio, se suele representar con el enlace a la palabra **nil** (nulo).

Un **puntero** es una variable cuyo valor es la dirección o posición de otra variable.

En las listas enlazadas no es necesario que los elementos de la lista sean almacenados en posiciones físicas adyacentes, ya que, el puntero indica dónde se encuentra el siguiente elemento de la lista.

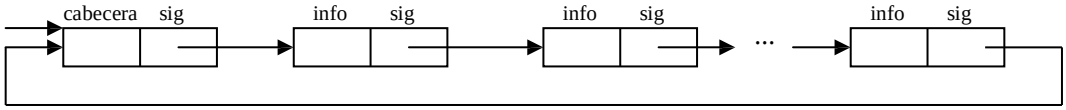


2.2. Listas circulares

Las listas simplemente enlazadas no permiten que a partir de un elemento cualquiera se pueda acceder a cualquiera de los elementos que le preceden. Para solucionar este problema, se pueden utilizar las listas circulares. En ellas, en lugar de almacenar un puntero nulo en el campo siguiente del último elemento de la lista, se hace que ése último elemento apunte al primer elemento de la lista.

En estas listas existe un nodo que recibe el nombre de **cabecera** porque indica el principio del recorrido de la lista circular. Este nodo tendrá una información especial diferente al resto de los nodos.

Una lista circular *vacía* será aquella que consta solamente del nodo cabecera.



2.3. Listas doblemente enlazadas

En una lista simplemente enlazada el recorrido sólo puede realizarse en un único sentido.

En las listas doblemente enlazadas cada nodo consta como mínimo de un campo de datos y de dos campos de enlace o punteros. Un campo puntero apuntará al siguiente elemento de la lista y el otro apuntará al elemento anterior. De esta manera, es posible recorrer estas listas en ambas direcciones.

3. PILAS

Una pila es un caso particular de lista en el que los elementos se añaden o eliminan sólo en un extremo. Se denomina **pila** ("stack") o **lista LIFO** ("Last Input, First Output") a una lista en la que las inserciones y eliminaciones se realizan sólo al principio de la lista. Es decir, cualquier elemento añadido pasa a ser el primero de la lista. Además, no se puede eliminar más que el elemento que ocupa el primer lugar de la lista en ese momento. Una pila se corresponde con la idea intuitiva de una pila de objetos, en

donde el último objeto colocado es el que se retira. Las pilas se utilizan en hardware y software para almacenar las direcciones de instrucciones desde las que se hacen llamadas a subrutinas.

Al extremo por donde se realizan las inserciones y eliminaciones se le denomina **cima** o tope ("top").

En principio, la pila estará vacía y el puntero cima de la pila estará a cero. Al meter un elemento en la pila, se incrementará el puntero en una unidad. Al extraer un elemento se decrementará.

Un *ejemplo de aplicación de una pila* es la comprobación de paréntesis de una expresión matemática, es decir, si existe igual número de paréntesis a la izquierda que a la derecha y que cada paréntesis de la derecha esté precedido por el correspondiente paréntesis de la izquierda.

4. COLAS

Las colas son otro tipo de estructura lineal de datos, similar a las pilas, diferenciándose de ellas en el modo de insertar y eliminar datos.

Se denomina **cola** a una lista en que las inserciones se realizan sólo en el final y sólo se puede acceder o eliminar en un instante dado el primer elemento de la lista (frente de la cola). Las colas se suelen denominar también **listas FIFO** ("First In, First Out").

Las colas se utilizan para almacenar datos que necesitan ser procesados según el orden de llegada. Se pueden utilizar en sistemas operativos para la gestión de trabajos no interactivos (procesamientos por lotes o "batch").

5. ÁRBOLES

El árbol es una estructura de datos fundamental en informática, muy utilizada en todos sus campos, porque se adapta a la representación natural de informaciones homogéneas organizadas y de gran comodidad y rapidez de manipulación. Se encuentra en todos los dominios, desde la pura algorítmica a la compilación o incluso dominios de la inteligencia artificial.

Las estructuras tipo árbol se usan principalmente para representar datos con una relación jerárquica entre sus elementos, como son los árboles genealógicos, tablas, etc.

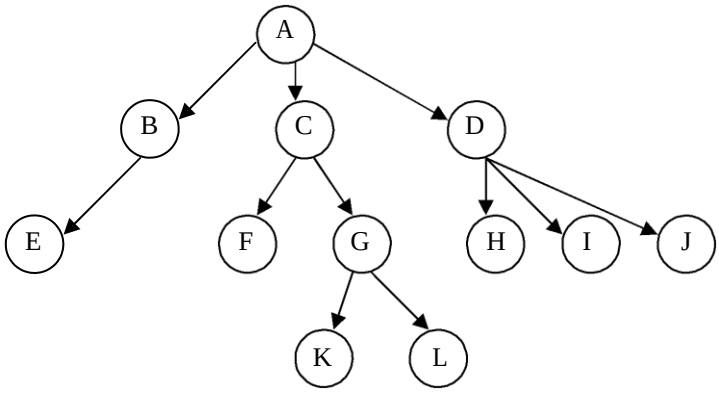
Un **árbol** *es una estructura de datos formada por elementos del mismo tipo, llamados* ***nodos****, relacionados de tal modo que*:

- *Existe un nodo especial denominado* ***raíz*** *del árbol*
- *Los nodos restantes se dividen en n 0 conjuntos distintos A_1, A_2, ..., A_n, cada uno de los cuales es, a su vez, un árbol. Éstos árboles se denominan* ***subárboles*** *del raíz.*

La definición de un árbol implica una estructura recursiva. Esto es, la definición del árbol se refiere a otros árboles. Un árbol con ningún nodo es un **árbol nulo**; no tiene raíz.

La representación y terminología de árboles se realiza con las típicas notaciones de las relaciones familiares en los árboles genealógicos: padre, hijo, hermano, ascendente, descendiente, etc.

Supongamos el siguiente árbol:



Las definiciones a tener en cuenta son:

- Se denomina **hijo** de un nodo a cada uno de los nodos que dependen de él. En la figura, F y G son hijos de C.
- Se dice también que C es el **padre** de estos nodos.
- Se denomina **grado** de un nodo al número de subárboles que sustenta. En la figura el grado de B es uno y el de D es tres.
- El **orden** de un árbol es el mayor de los grados de sus nodos.
- **Raíz** (A). Todos los árboles que no están vacíos tienen un único nodo raíz. Todos los demás elementos o nodos se derivan o descienden de él. El nodo raíz no tiene padre.
- **Nodo**. Son los vértices o elementos del árbol.
- Nodo terminal u **hoja**. Aquel nodo que no contiene ningún subárbol (E, F, K, L, H, I, J).
- Los nodos de un mismo padre se denominan **hermanos**.
- Los nodos con uno o más subárboles se llaman **nodos intermedios** o internos.
- Una colección de dos o más árboles se denomina **bosque**.
- Todos los nodos tienen un solo padre menos el raíz, que no tiene.
- Se denomina **camino** al enlace entre dos nodos consecutivos, y **rama** es el camino que termina en una hoja.
- Cada nodo tiene asociado un número de **nivel** que se determina por la longitud del camino desde la raíz al nodo especificado (A es de nivel 0; B, C, D tienen nivel 1; etc.).
- La altura o **profundidad** de un árbol es el número máximo de nodos de una rama. Equivale al nivel más alto de los nodos mas uno. El de ejemplo tiene profundidad 4.

5.1. Representación de árboles

Un árbol es una estructura dinámica. Su representación en el interior de un ordenador se realiza principalmente utilizando punteros. Cada nodo puede incluir varios punteros: uno para dirigirse al padre, y cada uno de los restantes para dirigirse a cada uno de los hijos. Esto permite moverse con gran facilidad dentro del árbol en cualquier dirección (hacia arriba o hacia abajo), pero presenta el inconveniente de que el número de punteros para cada nodo puede ser excesivamente grande, ya que, no es fijo. Normalmente se utiliza otra estructura con sólo tres punteros por nodo, uno para el padre, otro para el primer hijo, y el tercero para el siguiente hermano.

5.2. Árboles binarios

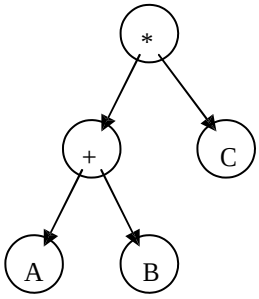
Un **árbol binario** es un conjunto finito de cero o más nodos tales que:

- Existe un nodo denominado raíz del árbol.
- Cada nodo puede tener 0, 1 o 2 subárboles conocidos como subárbol izquierdo y derecho.

Se dice que dos árboles binarios son **similares** si tienen la misma estructura y **equivalentes** si contienen la misma información.

Un árbol binario está **equilibrado** si las alturas de los dos subárboles de cada nodo del árbol se diferencian en una unidad como máximo.

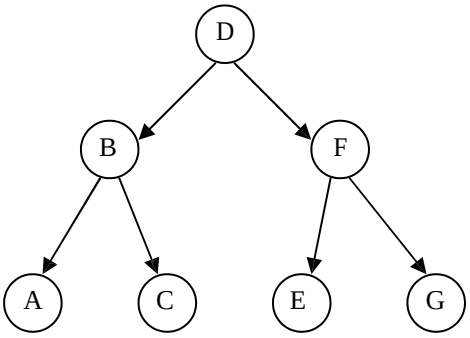
Los árboles binarios se pueden utilizar para almacenar expresiones en memoria. Por ejemplo, el siguiente árbol representa la expresión (A+B)*C:



5.2.1. Árboles binarios completos

Un árbol binario se llama **completo** si todos sus nodos tienen exactamente dos subárboles, excepto los nodos de los niveles más bajos que dos. Un árbol binario completo, tal que todos sus niveles estén llenos, se llama árbol binario **lleno**.

La **altura** de un árbol binario lleno de n nodos es $\mathbf{log_2(n+1)}$. A la inversa, el **número máximo de nodos** de un árbol binario de altura h, será $\mathbf{2^h-1}$. Todo esto se puede ver claramente en la siguiente figura:



Altura = 3 (número máximo de nodos de una rama)
Número máximo de nodos = $2^3-1 = 7$

5.2.2. Conversión de un árbol general en árbol binario

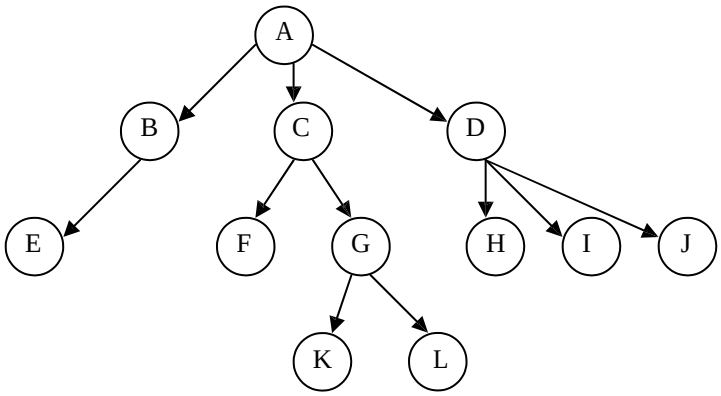
Los árboles binarios son más fáciles de programar que los generales. En éstos últimos, hay que deducir cuántas ramas o caminos cuelgan de un nodo en un momento dado. En los árboles binarios siempre cuelgan como máximo dos subárboles.

Existe una técnica para convertir un árbol general (A) a formato de binario (B); es el siguiente **algoritmo de transformación**:

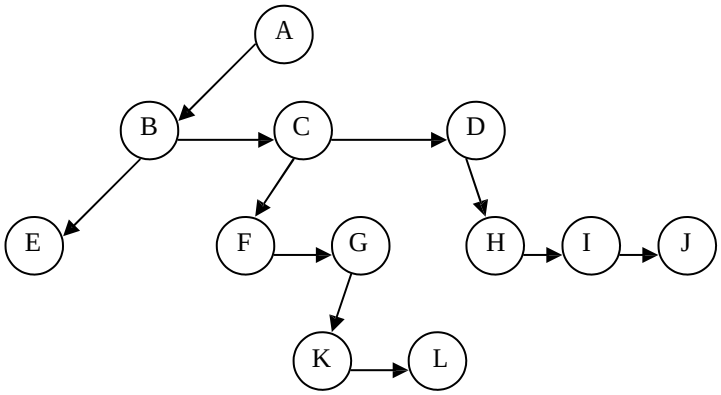
1. La raíz de B es la de A.
2.
   A) Enlazar el nodo raíz con el camino que conecta el nodo más a la izquierda (su hijo).
   B) Enlazar este nodo con los restantes descendientes del nodo raíz en un camino. Con esto se forma el nivel 1.
   C) A continuación, repetir los pasos A) y B) con todos los nodos del nivel 2, enlazando siempre en un mismo camino todos los hermanos. Repetirlos hasta el nivel más alto.
3. Girar el diagrama resultante 45 grados para diferenciar entre los subárboles izquierdo y derecho.

Ejemplo de aplicación del algoritmo:

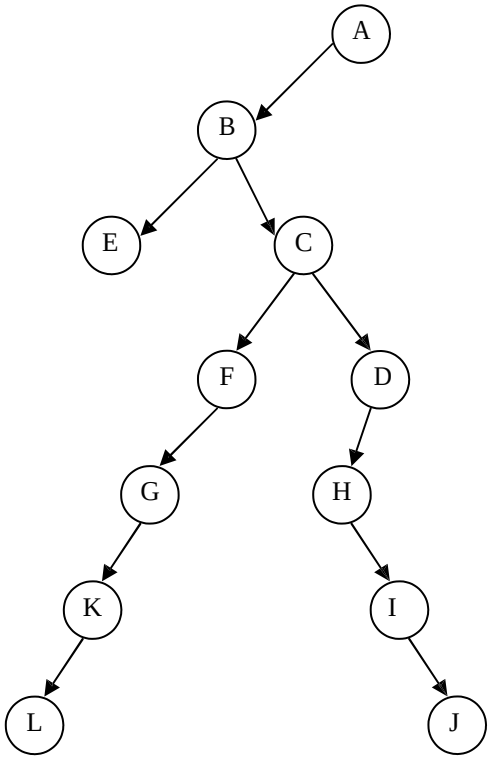
Considerando el siguiente árbol general, lo convertimos en árbol binario.



Aplicando el paso 2, queda:



Y, finalmente, al girar 45 grados, nos queda:



5.2.3. Representación de los árboles binarios

Lo habitual es utilizar punteros como en los árboles generales. Podríamos considerar el uso de una lista enlazada. Otra forma sería utilizando arrays.

5.2.4. Recorrido de un árbol binario

Se denomina **recorrido** de un árbol al proceso que permite acceder una sola vez a cada uno de los nodos del árbol.

Existen muchos modos de recorrer un árbol binario. En general, se agrupan en tres tipos de actividades comunes que son: *visitar el nodo raíz, recorrer el subárbol izquierdo y recorrer el subárbol derecho.*

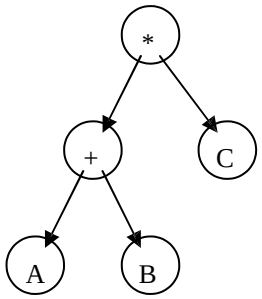
Estas tres acciones repartidas en diferentes órdenes proporcionan los diferentes recorridos del árbol. Los más frecuentes tienen siempre en común recorrer primero el subárbol izquierdo y luego el derecho. Estos **algoritmos** son los siguientes

- **Preorden**
  Visitar la raíz
  Recorrer el subárbol izquierdo en pre-orden
  Recorrer el subárbol derecho en pre-orden

- **Inorden**
  Recorrer el subárbol izquierdo en inorden
  Visitar la raíz
  Recorrer el subárbol derecho en inorden

- **Postorden**
  Recorrer el subárbol izquierdo en postorden
  Recorrer el subárbol derecho en postorden
  Visitar la raíz

Con el siguiente **ejemplo** vemos cómo se efectúan los recorridos:



En preorden se obtiene: * + A B C

En inorden se obtiene: A + B * C

En postorden se obtiene: A B + C *

Como ejemplo de algoritmo codificado en seudocódigo tenemos el siguiente que realiza un recorrido en inorden por un árbol binario (observar su recursividad):

```
Procedimiento Inorden  A  Arbol,  P  Posicion)
        Constantes

        Variables
                E  Elemento
                Q  Posicion
        Acciones
                Si No NodoNulo(P) Entonces
                        Q  HijoIzquierdo(A, P)
                        Inorden(A, Q)
                        E  Recupera(A, P)
                { Dar el tratamiento que se desee al elemento E }
                        Mientras No NodoNulo(Q) Hacer
                                Q  HermanoDerecho(A, Q)
                                Inorden(A, Q)
                        Fin Mientras
                Sino

                Fin Si
```

5.3. Árboles binarios de búsqueda

Un árbol binario de búsqueda se construirá teniendo en cuenta las siguientes premisas:

- El primer elemento se utiliza para crear el nodo raíz.
- Los valores del árbol deben ser tales que pueda existir un orden.
- En cualquier nodo todos los valores del subárbol izquierdo del nodo son menores o iguales al valor del nodo. De modo similar, todos los valores del subárbol derecho deben ser mayores que los valores del nodo.

Una característica de los árboles binarios de búsqueda es que no son únicos para los datos dados. La utilidad de este tipo de árbol se refiere a la eficiencia en la búsqueda de un nodo, similar a la búsqueda binaria. Los árboles binarios de búsqueda se utilizan típicamente como estructura de datos para la representación de conjuntos con operaciones de inserción, supresión y saber si un elemento dado es miembro o no del conjunto.

5.3.1. Búsqueda de un elemento

La búsqueda en un árbol binario ordenado es dicotómica, ya que, en cada examen de un nodo se elimina aquel de los subárboles que no contiene el valor buscado.

5.3.2. Inserción de un elemento

Para insertar un elemento, se ha de comprobar, en primer lugar, que el elemento no se encuentre ya en el árbol, ya que, en este caso no procede ser insertado. Si el elemento no existe, la inserción se realiza descendiendo por el árbol a partir del nodo raíz, dirigiéndose de izquierda a derecha de un nodo según que el valor a insertar sea inferior o superior al valor del campo a insertar. Cuando se alcanza el nodo del árbol en que se puede continuar, el nuevo elemento se engancha a la izquierda o derecha de este nodo en función de que su valor sea superior o inferior al del nodo alcanzado.

5.3.3. Eliminación de un elemento

La eliminación de un elemento debe conservar el orden de los elementos del árbol. Se consideran diferentes casos según la posición del elemento o nodo del árbol:

- Si el elemento es una hoja, se suprime sin más.
- Si el elemento no tiene mas que un descendiente, se sustituye por él.
- Si el elemento tiene dos descendientes, se sustituye por el elemento más a la derecha o más a la izquierda que permita conservar la ordenación.

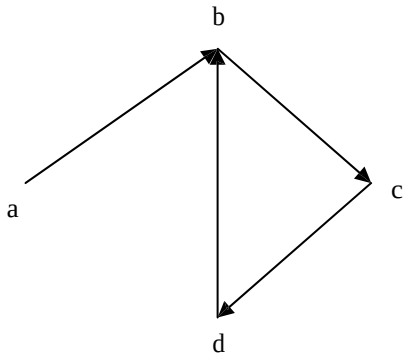
6. GRAFOS

Un **grafo** es una estructura de datos no lineal que tiene un gran número de aplicaciones.

Formalmente, un grafo es un conjunto de puntos y un conjunto de líneas, cada una de las cuales une un punto con otro. Los puntos se denominan **nodos** o vértices del grafo y las líneas se denominan aristas o **arcos**.

Un ejemplo de grafo es una red de carreteras.

Un grafo se dice que es **sencillo** si no tiene lazos, es decir, si no existen arcos entre el mismo elemento y no existe más que un arco para unir dos nodos. Un ejemplo de grafo sencillo es:



Cuando un grafo no es sencillo se denomina **múltiple**.

Un **camino** es una secuencia de uno o más arcos que conectan dos nodos. La **longitud** de un camino es el número de arcos que comprende.

En el grafo anterior una arista sería (a, b) y un camino C(b, d) = (b, c) (c, d).

Se dice que dos vértices son **adyacentes** si existe un arco que los une.

El **grado** de un nodo es el número de vértices que inciden en él.

Un grafo puede ser **dirigido**: cuando los vértices apuntan unos a otros; los arcos están dirigidos o tienen dirección. Un grafo es **no dirigido** cuando los vértices están relacionados, pero no se apuntan unos a otros; la dirección no es importante.

Un grafo es **conectado** cuando siempre existe un camino entre cualquier conjunto de vértices y **desconectado** cuando existen vértices que no están unidos por ningún camino.

6.1. Representación de grafos

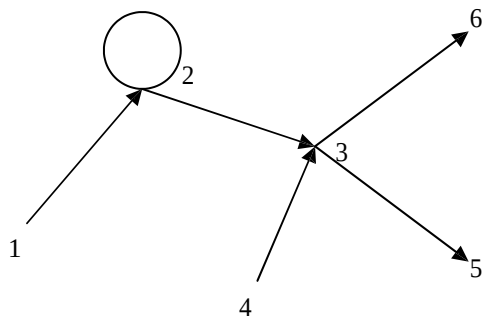
Un grafo puede estar representado por una matriz de adyacencia o por una lista de adyacencia.

Matriz de adyacencia

Si hay N nodos en un grafo la matriz de adyacencia será una tabla con N filas y N columnas.

El valor de la posición (i, j) de la tabla será 1, si existe un arco de un nodo a otro y cero si no existe. Si el grafo contiene factor de peso puede contener en la celda o posición (i, j), el peso sobre el arco y si no contiene peso o arco su valor será cero.

Ejemplo, dado el siguiente grafo dirigido, obtenemos su matriz de adyacencia:

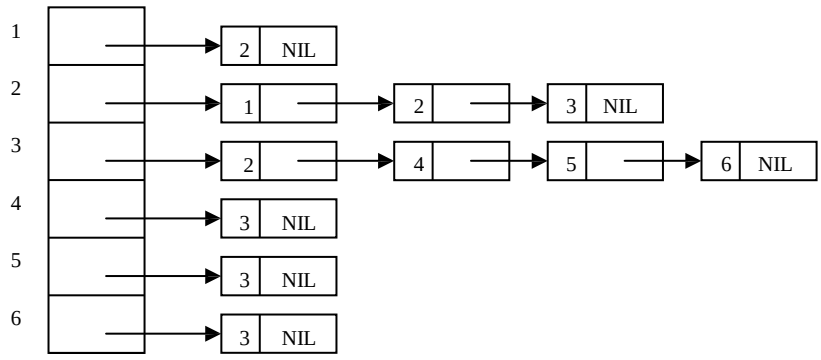


	j					
i	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	0	0	1	1
4	0	0	1	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Lista de adyacencia

Son listas enlazadas, una para cada nodo, conteniendo los nombres de los nodos a los que el nodo está conectado; las cabezas de cada una de estas listas se tienen en listas encadenadas o enlazadas.

Suponiendo que el grafo del ejemplo anterior fuese no dirigido, se obtendría:



7. BIBLIOGRAFÍA

Alberto Prieto
Introducción a la Informática
Mc Graw-Hill, 2ª edición, 1997

Alfonso Ureña López
Fundamentos de Informática
Ra-ma, 1997

Pascual Laporta, G.
Estructura de la Información
Mc Graw-Hill, 1992