TEMA 12

ORGANIZACIÓN LÓGICA DE LOS DATOS: ESTRUCTURAS DINÁMICAS

INDICE:

Realizado por Cayetano Borja Carrillo

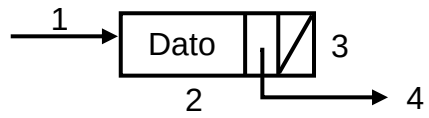
Tiempo de escritura: 2 horas

1. INTRODUCCIÓN

En informática, un dato es una unidad de información u objeto manipulable por un ordenador. Los datos pueden ser de tipo primitivo si almacenan un único valor simple (entero, flotante, carácter, etc.) o estructurado si se representan mediante una estructura de datos que puede contener varios valores.

Una estructura de datos es una forma de organizar una colección de datos con el fin de facilitar su acceso y manipulación. Las estructuras pueden ser estáticas o dinámicas, dependiendo de si el tamaño que ocupan en memoria es fijo o variable.

En este tema se desarrollan los principales conceptos de las estructuras de datos dinámicas, que son aquellas donde el tamaño que ocupan en memoria puede cambiar durante la ejecución del programa. Esto es posible porque se basan en nodos que almacenan los datos y una o varias referencias a otros nodos (punteros). Ejemplo de nodo:



1 – Puntero que contiene la dirección de memoria donde se encuentra el nodo.
2 – Nodo que almacena 3 campos: un dato y dos punteros.
3 – Puntero con valor nulo (NULL). No apunta a ningún lugar.
4 – Puntero que contiene la posición de memoria de otro nodo.

Se trata de un tema de gran importancia dentro del campo de estudio de la programación ya que, conocer cómo son estas estructuras, permite al desarrollador elegir la que mejor se adapte a sus necesidades.

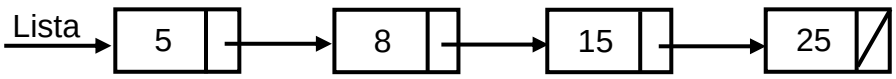
2. LISTA ENLAZADA

Una lista enlazada es una estructura de datos compuesta de un conjunto de nodos enlazados que mantienen una relación lineal (cada nodo tiene un antecesor y un sucesor, excepto el primero y el último) y secuencial (va pasando de un elemento a otro). Para saber dónde se encuentra el primer elemento de la lista, se utiliza un puntero inicial que llamaremos “Lista”.

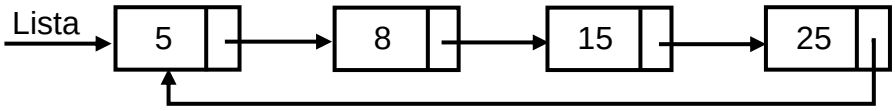
2.1. Tipos de listas enlazadas

Existen varios tipos de listas enlazadas, siendo las más comunes las siguientes:

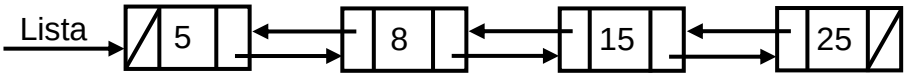
- Lista enlazada simple: Cada nodo contiene 2 campos, uno dato y un puntero que indica la ubicación del nodo sucesor. Ejemplo:



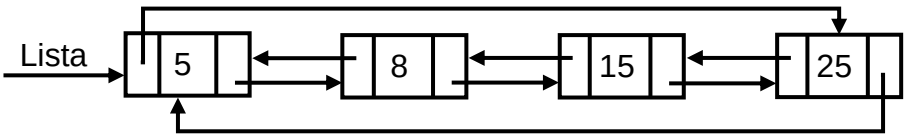
- Lista circular: Similar a una lista enlazada simple, con la diferencia de que el último nodo apunta al primer elemento de la lista. Ejemplo:



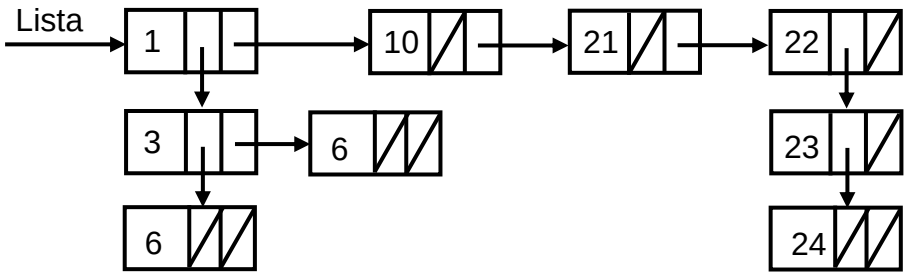
- Lista doblemente enlazada: Cada nodo contiene 2 punteros, uno que apunta al nodo sucesor y otro al antecesor. Ejemplo:



- Lista doblemente enlazada circular: Similar a la anterior, pero en este caso el nodo inicial apunta al último elemento y el último al primero. Ejemplo:



- Lista multinivel: Cada nodo tiene 2 punteros, uno que apunta al siguiente elemento de la lista de ese nivel y otro que apunta al primer elemento de una lista independiente (sublista) Ejemplo:



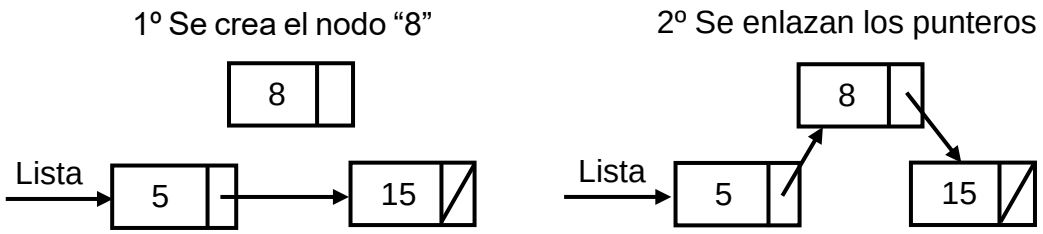
2.2. Operaciones con listas

Las operaciones que se pueden hacer con las listas son las siguientes:

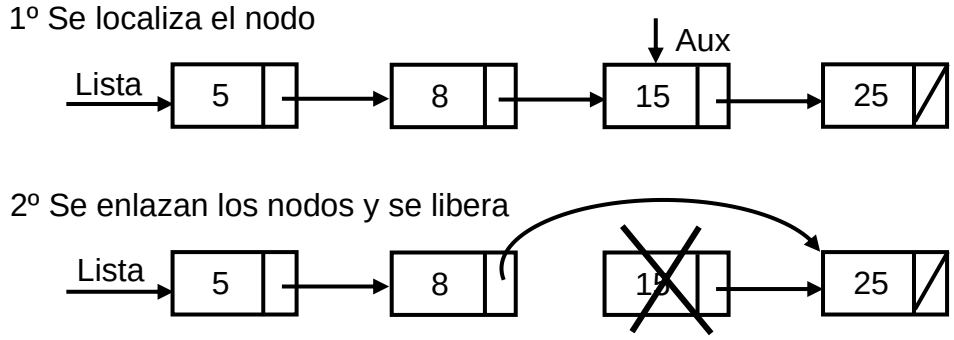
- Creación: Se crea la lista vacía, es decir, se crea un puntero inicial apuntando a nulo.

- Búsqueda: Se recorre la lista secuencialmente desde el principio hasta encontrar el elemento deseado o hasta llegar al final de la lista, en caso de que el elemento no exista.

- Inserción: Se crea el nuevo nodo, se busca la posición donde debe estar y se enlazan los punteros. Ejemplo: Insertar el elemento “8” en una lista ordenada cuyos elementos son “5” y “15”.



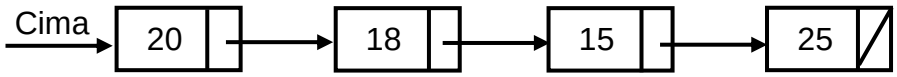
- Eliminación: Se busca el elemento que se quiere borrar y, si existe, se enlaza el puntero de su antecesor con su sucesor. Finalmente, se libera el nodo de memoria. Ejemplo: Dada la lista “5”, “8”, “15” y “25”, eliminar el elemento “15”.



- Vaciar: Se vacía la lista eliminando todos sus nodos uno a uno, hasta que el puntero inicial apunte a nulo.

3. PILA

Una pila es una estructura de datos de tipo LIFO (*Last In, First Out*) o último en entrar, primero en salir. Esto quiere decir que el último elemento que se introdujo en la pila estará en la cima o cabecera y el primero estará en el fondo de la pila. Ejemplo:



Observando la ilustración, se deduce que el elemento “25” es el más antiguo ya que está en el fondo y “20” el más reciente porque está en la cima.

3.1. Operaciones con pilas

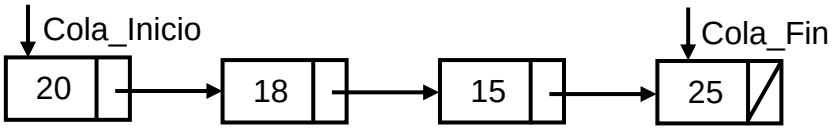
Las operaciones que se pueden hacer con las pilas son las siguientes:

- Creación: Se crea la pila vacía. El puntero inicial “Cima” apunta a nulo.
- Tamaño (*size*): Devuelve el número de elementos que hay en la pila.
- Apilar (*push*): Se introduce un nodo nuevo en la pila. El puntero inicial “cima” apuntará al nodo nuevo y éste al que estaba antes en la cabecera.
- Desapilar (*pop*): Lee y retira el elemento que hay en la cima.

- Leer último (*top*): Devuelve el valor del elemento que hay en la cima.
- Vacía: Devuelve "*true*" si la pila está vacía o "*false*" si contiene elementos.

4. COLA

Una cola es una estructura de datos de tipo FIFO (*First In, First Out*) o primero en entrar, primero en salir. Para implementarla se necesitan 2 punteros iniciales, uno para indicar el inicio de la cola y otro para indicar el fin. De esta forma, las inserciones se realizan por un extremo y las extracciones por el opuesto. Ejemplo:



Si se introduce un elemento nuevo, se inserta antes del "20" y el primer elemento en salir será el "25".

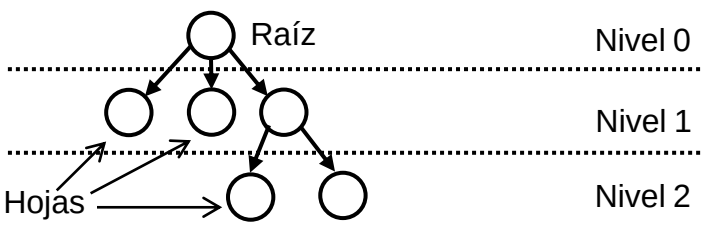
4.1. Operaciones con colas

Las operaciones que se pueden hacer con las colas son las siguientes:

- Creación: Se crea la cola vacía. Los 2 punteros iniciales apuntan a nulo.
- Frente: Devuelve el valor del elemento frontal, es decir, el valor del nodo que apunta "Cola_Fin".
- Desencolar: Se elimina el elemento frontal de la cola y se enlaza "Cola_Fin" con el que antes era el penúltimo elemento.
- Encolar: Se añade un nodo al final de la cola por "Cola_Inicio" y se enlazan los punteros.

5. ÁRBOL

Un árbol es una estructura no lineal cuyos nodos se organizan de forma jerárquica, como en un árbol genealógico. Cada elemento del árbol tiene un único antecesor (padre), pero puede tener varios sucesores (hijos). El primer elemento del árbol se llama "raíz", los nodos sin descendencia se llaman "hojas" y a cada generación en el árbol se le denomina "nivel". Ejemplo:

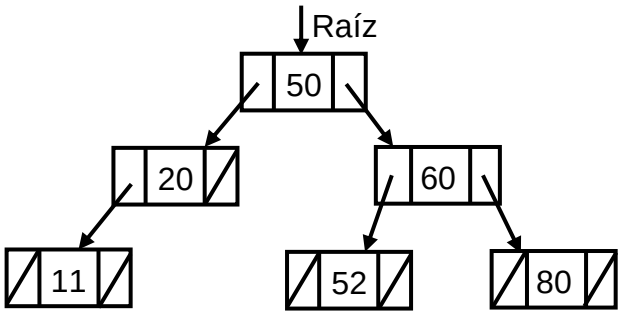


5.1. Tipos de árboles

Existen diferentes estructuras de tipo árbol, siendo algunos de ellos los siguientes:

Árbol binario de búsqueda (ABB)

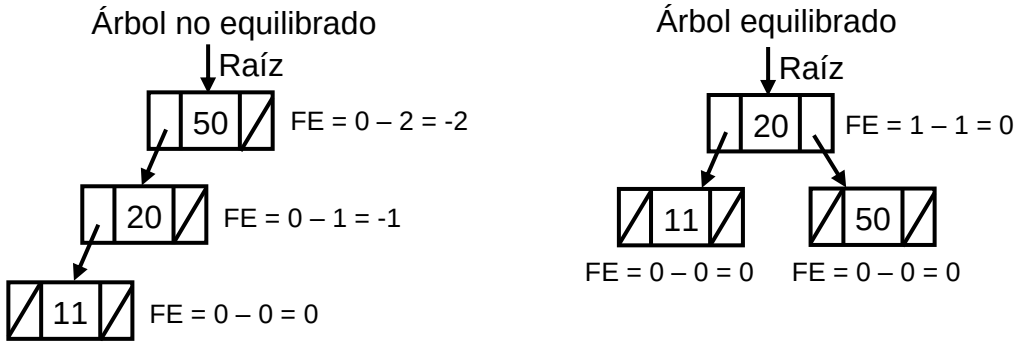
Se llama “binario” a este tipo de árbol porque sus elementos pueden tener como máximo 2 descendientes y “de búsqueda” porque su organización ordenada permite realizar búsquedas eficientes. Para mantener la información ordenada, los descendientes del subárbol izquierdo de cualquier nodo deben contener valores menores al suyo y los del subárbol derecho deben tener valores mayores. Ejemplo:



Como se puede observar, los elementos a la izquierda de “50” son menores y a la derecha mayores.

Árbol Addelson-Velskii y Landis (árbol AVL)

Un árbol AVL es un ABB equilibrado, es decir, es un árbol cuyo factor de equilibrio o FE (Altura subárbol derecho – Altura subárbol izquierdo) debe ser igual a -1, 0 o 1 en todos sus nodos. Ejemplo:



Al estar siempre en equilibrio, las búsquedas en un árbol AVL suelen ser más rápidas que en un ABB, pero presenta el inconveniente de que, si se realiza una inserción o una eliminación y el árbol se desequilibra, hay que reequilibrarlo aplicando rotaciones de los nodos.

Otros árboles

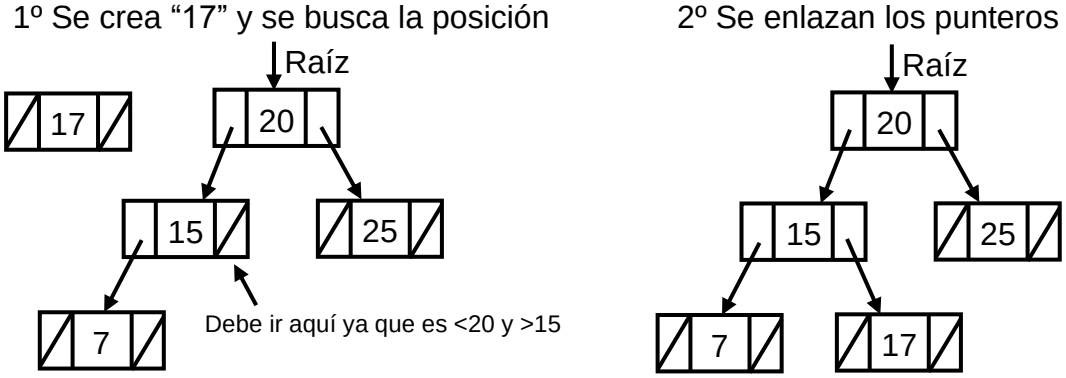
Además del árbol AVL y del ABB, existen otros tipos de árboles como:

- Árbol rojinegro: Se trata de un ABB equilibrado como el árbol AVL, pero obtiene su equilibrio de una forma distinta y menos rígida.
- Árbol B y B+: Otro tipo de árboles cuyos nodos pueden tener más de 2 descendientes.

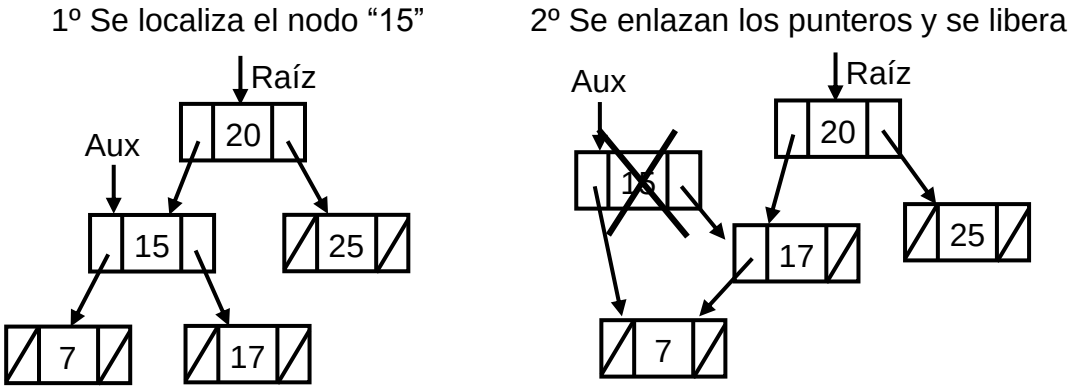
5.2. Operaciones con árboles

Las operaciones que se pueden hacer con los árboles son las siguientes:

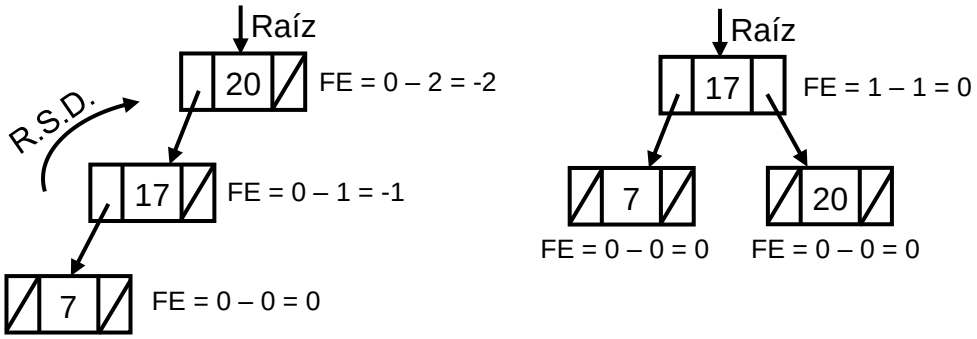
- Creación: Se crea el árbol vacío. El puntero “raíz” apunta a nulo.
- Búsqueda: Para buscar un elemento, se comprueba desde la raíz si el elemento que se busca es mayor o menor que el nodo donde estamos ubicados. Si es mayor, se realiza la misma operación con el nodo de su derecha y si es menor con el de su izquierda. Esta operación se repite recursivamente hasta llegar al nodo deseado o a nulo, en caso de no existir.
- Inserción: Se crea el nuevo nodo, se busca dónde debe ir y se enlazan los punteros. Ejemplo: Dado el siguiente árbol, insertar el elemento “17”.



- Eliminación: Se busca el elemento que se quiere borrar y, si existe, se enlazan los punteros y se libera el nodo de memoria. Ejemplo: Supongamos que en el árbol del ejemplo anterior se quiere eliminar el elemento “15”.



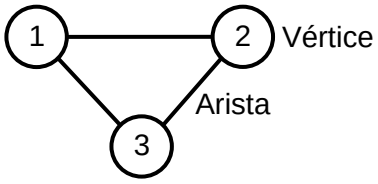
- Rotaciones (Sólo para árboles AVL): Si al insertar o eliminar un nodo, el árbol se desequilibra, se tiene que reequilibrar mediante rotaciones para que siga siendo AVL. Estas rotaciones pueden ser simples o dobles que, a su vez, pueden ser a la izquierda o a la derecha. Ejemplo: Supongamos que en el árbol resultante del ejemplo anterior se quiere eliminar "25", el árbol se desequilibraría porque el nodo raíz tendría un FE=-2. Para reequilibrarlo se aplica una rotación simple a la derecha o R.S.D.



6. GRAFO

Los árboles representan estructuras jerárquicas con la limitación de no permitir referencias circulares. Esto puede suponer un problema para representar determinados tipos de información como, por ejemplo, una red de carreteras. Esta limitación se soluciona con los grafos.

Un grafo se define como una estructura G = (V, A), donde "V" es un conjunto de nodos llamados vértices y "A" las aristas que unen los vértices. Ejemplo:



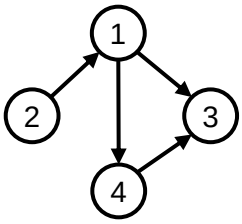
6.1. Tipos de grafos

Existen diferentes tipos de grafos cuya clasificación puede realizarse atendiendo a una serie de criterios como los siguientes:

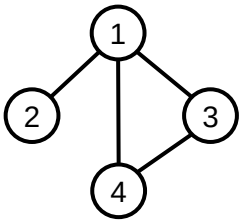
Dependiendo del sentido de sus aristas, un grafo puede ser:

- Dirigido: Las aristas el grafo tienen un sentido definido. Se representan con una flecha indicando el sentido.

- No dirigido: Las aristas del grafo son bidireccionales. Equivale a 2 aristas dirigidas, una para cada sentido.



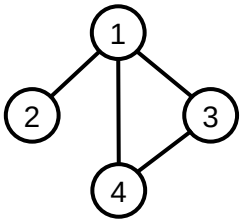
Grafo dirigido



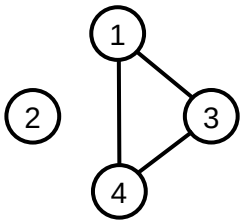
Grafo no dirigido

Dependiendo del número de aristas, un grafo puede ser:

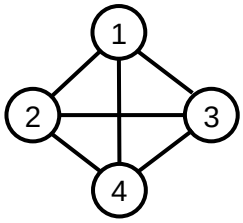
- Conexo: Es aquel donde existe un posible camino para cualquier par de vértices.
- Disconexo: Un grafo es disconexo si existe un vértice o grupo de vértices incomunicados con el resto.
- Completo: Un grafo no dirigido es completo si todos sus vértices están conectados entre sí.



Grafo conexo



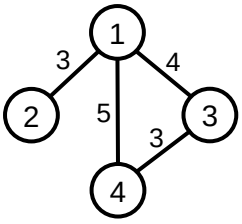
Grafo disconexo



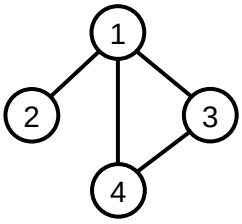
Grafo no dirigido

Dependiendo del peso de las aristas, un grafo puede ser:

- Ponderado: Un grafo es ponderado cuando cada arista tiene un valor asignado llamado peso. Esto es útil, por ejemplo, en una red de carreteras para referirse a los kilómetros.
- No ponderado: Cuando las aristas no tienen un valor establecido.



Grafo ponderado



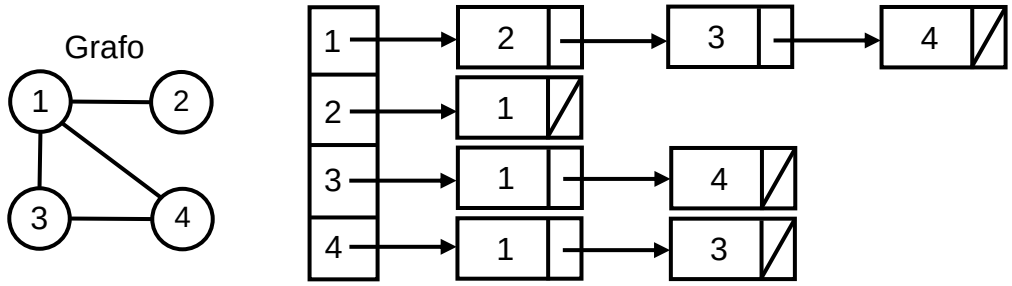
Grafo no ponderado

6.2. Formas de representar un grafo

Existen 3 formas de representar un grafo, mediante una estructura estática llamada matriz de adyacencias y mediante 2 estructuras dinámicas llamadas lista de adyacencias y multilista de adyacencias. Como este tema trata de las estructuras dinámicas, se van a describir las 2 últimas.

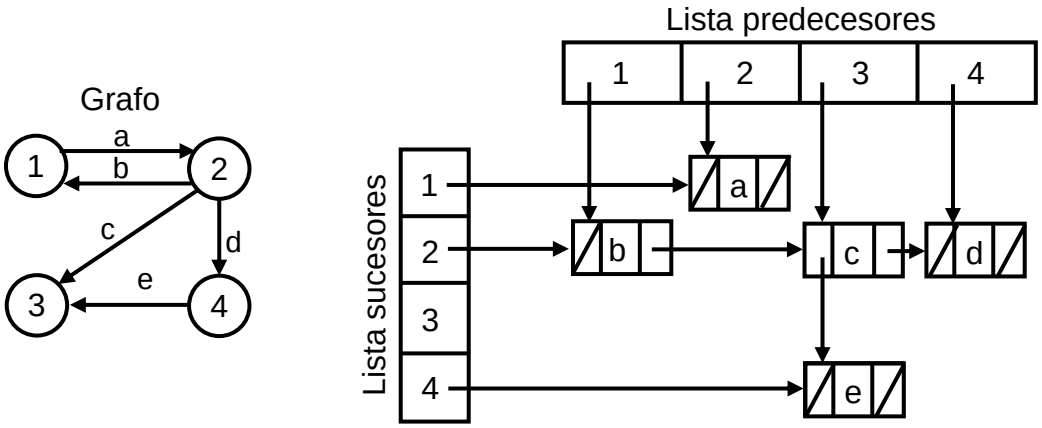
Grafo mediante una lista de adyacencias

Consiste en una lista donde cada elemento representa un vértice del grafo y de cada vértice sale otra lista con sus vértices adyacentes (aristas). Ejemplo:



Grafo mediante una multilista de adyacencias

En este caso se utilizan 2 listas, una para indicar los vértices sucesores y otra para los predecesores, es decir, una lista para asignar a cada vértice las aristas que salen de él y otra para asignar las aristas que le llegan. Esta estructura sólo tiene sentido en grafos dirigidos.



6.3. Operaciones con grafos

Las operaciones que se pueden hacer con los grafos son las siguientes:

- Creación: Crea un grafo vacío.
- Añadir vértice: Se añade un nodo al grafo.
- Añadir arista: Se añade un nuevo arco entre dos nodos.
- Borrar vértice o arista: Se elimina un vértice o arco y se reajustan los punteros si es necesario.
- Vacío: Devuelve "*true*" si el grafo está vacío y "*false*" si contiene vértices.
- Adyacentes: Devuelve "*true*" si los 2 nodos de entrada son adyacentes y "*false*" en caso contrario.

7. CONCLUSIÓN

Durante el desarrollo de un *software*, el programador va declarando variables y diseñado estructuras capaces de almacenar información según los requerimientos del programa que está en desarrollo.

En este tema se han abordado las estructuras de datos dinámicas, que son aquellas donde el tamaño que ocupan puede cambiar durante la ejecución del programa. Esto es posible porque no se basan en estructuras fijas, sino en nodos que contienen los datos y referencias a otros nodos.

Existen varios tipos de estructuras dinámicas como las listas enlazadas, las pilas, las colas, los árboles y los grafos. Cada una de estas estructuras está diseñada para almacenar la información de una forma determinada.

Conocer la organización lógica de cada una de estas estructuras y sus operaciones es fundamental para desarrollar un *software* de calidad, eficiente y sin errores.

8. BIBLIOGRAFÍA

- López Ureña, L. A. et al. (1997). *Fundamentos de Informática (1ª ed.)*. Ra-ma.
- Prieto Espinosa, A. et al. (2006). *Introducción a la informática (4ª ed.)*. McGraw-Hill.
- Brookshear, J. G. (2012). *Introducción a la computación (11ª ed.)*. Pearson Educación.
- Joyanes Aguilar, L. (2020). *Fundamentos de programación (5ª ed.)*. McGraw-Hill.

9. NORMATIVA

Para el desarrollo de este tema, se ha tenido en cuenta la siguiente normativa, donde se especifican los contenidos, competencias y criterios de evaluación de los Ciclos Formativos y Bachillerato en Andalucía:

- Orden 16 de junio de 2011 (DAW/DAM). La parte correspondiente a los módulos “Entornos de Desarrollo” y “Programación”.
- Instrucción 13/2022 (Bachillerato). La parte correspondiente a la asignatura “Tecnologías de la Información y Comunicación”