

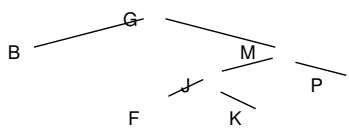
## GRAFOS

Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de cruce y avance es un grafo acíclico dirigido.	V
La representación de un grafo mediante una lista de adyacencia, siempre va a ser mejor tanto espacial como temporalmente que la representación mediante una matriz de adyacencia.	F
Los arcos de cruce de un recorrido en profundidad de un grafo dirigido, son los que van de un vértice a un descendiente propio del bosque extendido y no son "arcos de árbol".	F
Al clasificar las aristas de un grafo no dirigido en un recorrido en profundidad, sólo existen aristas de árbol y de retroceso.	V
Los árboles extendidos de un grafo tienen que ser necesariamente un árbol binario.	F
Al representar un grafo de N vértices y K aristas con una lista de adyacencia, la operación que halla la adyacencia de salida de un vértice, tiene una complejidad de $O(N)$ .	V
Sea $G = (V, A)$ un grafo dirigido. Diremos que $G' = (V', A')$ es un árbol extendido de $G \iff V' = V, A' \subseteq A$ , para todo $v \in V' \rightarrow \text{grado } \epsilon(v) \leq 1$	V
En un grafo no dirigido de 'n' vértices pueden existir infinitas aristas.	F
Dado un grafo dirigido, siempre se cumple que $\text{Adyacencia\_de\_Salida}(x) \cap \text{Adyacencia\_de\_Entrada}(x) = \emptyset$ , donde x es un vértice del grafo.	F
La adyacencia de un vértice w en un grafo no dirigido es el conjunto de aristas que tienen como origen o destino a w.	F
Ciclo es cualquier camino en el que el vértice primero y último coinciden.	F
Al representar un grafo no ponderado de N vértices y K aristas con una matriz de adyacencia, la operación de búsqueda de una arista tiene una complejidad de $O(N)$ .	F
En un multigrafo pueden existir infinitas aristas para un número 'n' de vertices.	V
En un grafo dirigido pueden existir infinitas aristas para un número 'n' de verices.	F
Un grafo no dirigido de n vértices es un árbol libre si está libre de ciclos y tiene 'n-1' aristas.	V
Un bosque extendido en profundidad de un grafo no dirigido es un grafo acíclico.	V
La siguiente secuencia de nodos en un grafo es un ciclo: 1,2,3,2,1.	F
Un bosque extendido en profundidad de un grafo dirigido también es un grafo acíclico dirigido.	V

## ÁRBOLES GENERAL

El grado de un nodo es el máximo número de etiquetas de dicho nodo más uno.	V
El grado de un árbol es el número mínimo de hijos que puede tener sus subárboles	F
Un bosque de grado 3 se puede representar como un árbol binario	V
La profundidad de un subárbol es la longitud del único camino desde la raíz a dicho subárbol.	V
El número de nodos de un árbol Fibonacci de altura h es el máximo número de nodos que puede tener un árbol binario de altura h para que sea AVL.	F
En un árbol binario enhebrado todas las hojas tienen 2 hebras.	F
El grado de un árbol es el máximo nivel de los nodos de un árbol.	F
En un árbol binario enhebrado todos los nodos tienen 2 ó más hebras ó 2 hijos ó 1 hijo y 1 hebra.	F
Sólo existen tres formas de recorrido en profundidad en un árbol binario: preorden, inorden y postorden.	F
La estructura de datos árbol aparece porque los elementos que lo constituyen mantienen una estructura jerárquica, obtenida a partir de estructuras lineales, al eliminar el requisito de que cada elemento tiene como máximo un sucesor y un predecesor.	F
Un bosque se puede representar con un único árbol binario.	V
Existe al menos un árbol, que representa los siguiente recorridos: inorden=XYZT, niveles=XTYZ.	F
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo.	F
Dado un único recorrido de cualquier árbol, es posible reconstruir dicho árbol.	F
El grado de un nodo es el número de ítems asociados a dicho nodo.	F
Un árbol de Fibonacci siempre está balanceado respecto a la altura.	V
Dado un único recorrido de un árbol lleno, es posible reconstruir dicho árbol.	V
Un árbol de Fibonacci es un árbol balanceado con respecto a la altura.	V
Los árboles generales también se les llaman multicamino de búsqueda.	F
Un árbol completo es un árbol completamente equilibrado.	F
El grado de un árbol es el grado mínimo de todos los nodos de ese árbol.	F
Un árbol binario lleno es un árbol binario completo.	V
En un árbol binario enhebrado, el primer modo del recorrido en inorden no tiene hebra izquierda.	V

## ÁRBOLES AB Y ABB

El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	V
Un árbol binario lleno cumple las condiciones para ser también árbol Leftlist.	V
Un árbol binario completo, el camino mínimo de la raíz es igual a la altura del árbol	F
En un árbol binario, el camino mínimo de la raíz es igual a la altura del árbol.	F
Si en un árbol binario representado secuencialmente tenemos el nodo padre en la posición 5, sus hijos izquierdo y derecho se encuentran, respectivamente, en las posiciones 6 y 7.	F
El menor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja.	F
En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el mayor del subárbol de la izquierda o por el menor del subárbol de la derecha.	V
Cuando realizamos un recorrido por niveles en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor.	F
Cuando realizamos un recorrido en inorden en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor.	V
En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el menor del subárbol de la izquierda o por el mayor del subárbol de la derecha.	F
Cuando realizamos un recorrido en preorden en un ABB las etiquetas aparecen ordenadas de menor a mayor.	F
El siguiente árbol es binario de búsqueda, usando el orden alfabético como sistema de ordenación.	F
 <pre> graph TD     G --- B     G --- M     M --- J     M --- P     J --- F     J --- K </pre>	
En un árbol binario cada elemento puede tener como máximo dos predecesores.	F
Existe un único árbol binario completo que se puede construir a partir del recorrido en postorden.	V

## ÁRBOLES AVL

El borrado de un árbol AVL puede requerir una rotación en todos los nodos del camino de búsqueda.	V
En el borrado del AVL, la altura del árbol decrece siempre tras realizar una rotación simple.	F
Dado un recorrido en postorden se puede reconstruir más de un árbol AVL	F
En un árbol AVL, al realizar una inserción de una sola clave se puede producir como máximo una rotación	F
En la inserción de un AVL el equilibrado se realiza de las hojas hacia la raíz.	V
El factor de equilibrio en los nodos de un árbol AVL tiene que ser cero para que no haya que reequilibrar el árbol en una operación de inserción o borrado.	F
Las rotaciones que hay que realizar en los árboles AVL para mantenerlos balanceados tiene un coste temporal lineal con respecto al número de ítems del árbol.	F
Cuando se realiza un borrado en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	F
Los árboles AVL son aquellos en los que el número de elementos en los subárboles izquierdo y derecho difieren como mucho en 1.	F
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	V
Dado un recorrido en preorden (RID) de un árbol AVL es posible reconstruir un único árbol AVL.	V
Para realizar la inserción de un elemento en un árbol AVL que sea el máximo o el mínimo, se realizará como máximo una rotación doble.	F

## ÁRBOLES 2-3

El número mínimo de elementos que se puede almacenar en un árbol 2-3 de altura h coincide con el número de elementos que hay en un árbol binario completo de altura h.	F
En operación de Rotación en un árbol 2-3 se da cuando el hermano del nodo donde se efectúa el borrado es del tipo 2-nodo	F
Los nodos de grado 0 de un árbol 2-3 pueden estar en distintos niveles del árbol.	F
El grado del árbol 2-3 es 2	F
Dado un árbol 2-3 con n ítems con todos sus nodos del tipo 2-nodo: la complejidad de la operación de búsqueda de ítems de $O(\log_2 n)$ .	V
Dado un árbol 2-3 de altura h con n ítems: $2^h - 1 \leq n \leq 3^h - 1$ .	V
Un árbol 2-3 es un árbol 2-camino de búsqueda.	F
El árbol 2-3 es un árbol B m-camino de búsqueda con m=2.	F
El mínimo número de ítems que se puede almacenar en un árbol 2-3 de altura h es menor que el número de ítems que hay en un árbol binario lleno de altura h.	F
El número mínimo de elementos que se puede almacenar en un árbol 2-3 de altura h coincide con el número de elementos que hay en un árbol binario lleno de altura h.	V
El número mínimo de elementos que se pueden almacenar en un árbol 2-3 de altura h es $3^h - 1$ .	F
El árbol 2-3 es un árbol B m-camino de búsqueda con m=3.	V
El número máximo de elementos que se puede almacenar en un árbol 2-3 de altura h es $3h - 1$ .	V
El máximo grado de cualquier nodo de un árbol 2-3 es 3.	V
Dado un árbol 2-3 de altura h con n ítems: $3^h - 1 < n < 2^h - 1$ .	F

## ÁRBOLES 2-3-4

En el borrado del árbol 2-3-4 sólo se reduce la altura del árbol cuando p,q y r son 2-nodo.	V
Un árbol 2-3-4 cumple las condiciones para ser también un árbol Leftlist.	F
En la operación de inserción en un árbol 2-3-4 las reestructuraciones se realizan desde las hojas hacia la raíz	F
El número de elementos que hay en un árbol 2-3-4 de altura h está comprendido entre $2^h - 1$ y $4^h - 1$ .	V
En la operación de inserción en un árbol 2-3-4 sólo se divide la raíz si ésta es un 3-nodo.	F
En la operación de borrado en un árbol 2-3-4 siempre que sea 2-nodo hay que hacer una COMBINACION o una ROTACION	V
En un árbol 2-3-4 con n elementos, la altura de dicho árbol se encuentra entre $\log_4(n+1)$ y $\log_2(n+1)$ .	V
En un árbol 2-3-4 las inserciones siempre se realizan en las hojas.	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles.	V
Un árbol 2-3-4 es un árbol binario completo.	F
En un árbol 2-3-4 todas las hojas están al mismo nivel.	V
El árbol 2-3-4 tiene como mínimo una clave en cada nodo.	V
La operación de inserción en un árbol 2-3-4 requiere una reestructuración del árbol en el camino de vuelta atrás de las hojas a la raíz.	F
Para que decrezca la altura de un árbol 2-3-4 en una operación de borrado, el nodo raíz y sus hijos tienen que ser 2-nodo.	V
Un árbol 2-3-4 es un árbol 4-camino de búsqueda.	V
El árbol 2-3-4 tiene como mínimo dos claves en cada nodo.	F
Las operaciones de transformación en un árbol 2-3-4 se reducen a cambios de colores o rotaciones en un árbol Rojo-Negro.	V
En un árbol 2-3-4 de altura h, el máximo número de nodos se da cuando todos los nodos son de tipo 2-nodo.	F
Al borrar un elemento en un árbol 2-3-4 se puede realizar una operación de DIVIDERAIZ.	F

## ÁRBOLES ROJOS-NEGROS

En un árbol Rojo-Negro el algoritmo de búsqueda de una etiqueta es el mismo que el empleado en un árbol binario de búsqueda.	V
Un árbol Rojo-Negro es un ABB que representa a un árbol 2-3.	F
En un árbol Rojo-Negro ningún camino desde la raíz a las hojas tiene dos o más hijos negros consecutivos.	F
Las hojas en un árbol Rojo-Negro están enlazadas a sus padres siempre en color rojo.	F
Un árbol Rojo-Negro es un árbol balanceado respecto al número de hijos negros que hay desde la raíz hasta cada una de las hojas.	V
Se puede aplicar exactamente el mismo algoritmo de búsqueda de una etiqueta en un árbol binario de búsqueda que en un árbol Rojo-Negro.	V
Un árbol Rojo-Negro es un árbol binario balanceado respecto la altura.	F
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3-4.	V
Un árbol Rojo-Negro es un árbol m-camino de búsqueda con m=2.	V
El recorrido inorden en un árbol Rojo-Negro es el mismo que el realizado sobre un árbol binario.	V
En un árbol Rojo-Negro la búsqueda de una etiqueta dependerá de los colores de los hijos de cada nodo.	F
La inserción de una etiqueta en un árbol Rojo-Negro se efectuará creando un hijo de color negro.	F
Un árbol Rojo-Negro es un árbol balanceado respecto a la altura.	F
En un árbol Rojo-Negro el número total de hijos negros en cualquier camino desde la raíz a las hojas es siempre el mismo.	V

## ÁRBOLES B

El árbol B m-camino de búsqueda con altura 'k' tiene como máximo $m^k - 1$ claves.	V
El árbol B m-camino de búsqueda tiene como máximo $2^m - 1$ claves.	F
En el árbol B, la inserción de una clave sólo se puede realizar en las hojas del árbol	V
En un árbol B m-camino de búsqueda con m=16, en cualquier nodo excepto la raíz hay 7 ítems como mínimo	V
En un árbol B tiene que haber el mismo número de claves en el hijo izquierdo de la raíz que en el hijo derecho	F
El nodo de un árbol B m-camino de búsqueda con m=4 puede tener como máximo 4 hijos no vacíos.	V
La altura del árbol B m-camino de búsqueda con m>5 es menor o igual que la del árbol 2-3, o 2-3-4 para el mismo número de etiquetas.	V
Un árbol B siempre se puede transformar en un árbol binario de búsqueda completo.	V
En un árbol B m-camino de búsqueda con m=18; en cualquier nodo excepto la raíz hay 8 ítems como mínimo.	V
Un árbol B m-camino de búsqueda puede tener un nodo hoja en el nivel 3 y otro nodo hoja en el nivel 4.	F
En el árbol B m-camino de búsqueda, si un nodo tiene 'x' claves, entonces 'm' es menor de 'x'.	F
El árbol m-camino de búsqueda lleno con altura 'k' tiene $2^k - 1$ claves.	F
El nodo de un árbol B m-camino de búsqueda puede tener como máximo m ítems.	F
El nodo de un árbol B m-camino de búsqueda puede tener como máximo m hijos no vacíos.	V
La raíz del árbol B m-camino de búsqueda siempre tiene al menos m/2 claves o etiquetas.	F
El árbol B m-camino de búsqueda si tiene un nodo con 'x' claves, entonces ese nodo tendrá 'x+1' hijos.	V
En un árbol B m-camino de búsqueda con m=16: en cualquier nodo excepto la raíz hay 8 ítems como mínimo.	F
El árbol B m-camino de búsqueda con altura 'k' tiene como máximo $m^{k-1}$ claves.	F

## CONJUNTOS GENERAL

La altura máxima de un árbol de búsqueda digital es 'n+1', siendo n el número de bits de la clave.	V
La representación de conjuntos mediante listas el espacio es proporcional al tamaño del conjunto representado.	V
El proceso de búsqueda en un árbol digital, la elección de la siguiente rama a explorar viene determinada por la longitud de la clave buscada.	F
En la representación de conjuntos mediante lista el espacio es proporcional al tamaño del conjunto universal.	F
La mejor representación de los conjuntos siempre es el vector de bits porque es la representación más eficiente espacialmente.	F
En un árbol de búsqueda digital, la elección de la siguiente rama del árbol a explorar en un proceso de búsqueda, viene determinada por la longitud total de la clave buscada.	F

## TRIE

La altura de un Trie con nodos terminales será como mínimo la longitud de la cadena más larga almacenada.	F
La altura de un Trie en su peor caso vendrá determinada por la cadena de menor longitud almacenada en el árbol.	F
En un proceso de búsqueda de una cadena en un trie la elección de la siguiente rama a explorar viene determinada por la siguiente letra de palabra.	V
La altura de un Trie vendrá determinada por la cadena de mayor longitud almacenada en el árbol.	V
La representación del nodo de un trie utilizando un vector de punteros siempre es más eficiente espacialmente que utilizar una lista de punteros.	F

## DICCIONARIO

En el TAD Diccionario con dispersión cerrada, con función de redispersión $h_i(x) = (h_{i-1}(x) + C) \text{ MOD } B$ . Dos claves sinónimas (x,y) tendrán la misma secuencia de intentos, es decir, $h_i(x) = h_i(y)$ para cualquier valor de C y B.	V
En el TAD Diccionario con dispersión cerrada, con función de redispersión $h_i(x) = (H(x) + k(x)*i) \text{ MOD } B$ , "B" y "k(x)" no han de tener factores primos comunes mayores que uno, para que se busque una casilla libre por toda la tabla.	V
En el TAD Diccionario con dispersión cerrada, con función de redispersión $h_i(x) = (H(x) + C*i) \text{ MOD } B$ , "B" y "C" han de tener factores primos comunes mayores que uno, para que se busque una casilla libre por toda la tabla.	F
El TAD Diccionario es un subtipo del TAD Conjunto.	V
En el TAD Diccionario con dispersión abierta, la operación de búsqueda de una clave tiene una complejidad $O(L)$ , con L=longitud de la lista de claves sinónimas colisionadas.	V
En el TAD Diccionario con dispersión cerrada, cualquier estrategia de redispersión cuyo siguiente intento esté sólo en función del anterior, producirá amontonamiento.	V
En el TAD Diccionario con dispersión cerrada, los elementos se almacenan en una tabla de tamaño fijo.	V
En el TAD Diccionario con dispersión abierta, para evitar el problema del amontonamiento es aconsejable que el tamaño de la tabla sea un número primo o que no tenga factores primos menores que 20.	F
En el TAD Diccionario con dispersión cerrada, con función de redispersión " $h_i(x)=(H(x) + k(x)*i) \text{ MOD } B$ ", con B=6 se puede dar la situación de que en una búsqueda no se acceda a todas las posiciones de la tabla.	V

## TABLA DE DISPERSIÓN

En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $h_i(14) = (28 + 2^i) \text{ MOD } 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.	F
En la dispersión cerrada se pueden producir colisiones entre claves sinónimas y no sinónimas.	V
Sea una tabla de dispersión cerrada con función de dispersión $H(x) = x \text{ MOD } B$ , con $B=100$ y "x" un número natural entre 1 y 2000. Sólo hay un valor de "x" que haga $H(x)=4$ .	F
En la dispersión abierta, el número de intentos a realizar en la búsqueda sin éxito siempre es mayor o igual que en el borrado.	V
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $h_i(14) = (28 + 7^i) \text{ MOD } 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.	V
Sea una tabla de dispersión cerrada con estrategia de redispersión $h_i(x) = (H(x) + C^i) \text{ MOD } B$ , con $B=1000$ y $C=74$ . Para cualquier clave 'x' se recorrerán todas las posiciones de la tabla buscando una posición libre.	F
El proceso de dispersión es opuesto al de ordenación.	V
La dispersión cerrada con estrategia de redispersión aleatoria tiene la siguiente función de redispersión: $h_i(x) = (h_{i-1}(x) + C) \text{ MOD } B$ .	V
En la dispersión abierta sólo se producen colisiones entre claves sinónimas.	V

## COLA DE PRIORIDAD

El TAD Cola de Prioridad Doble en el que no se permiten elementos repetidos, representado por una lista desordenada, tendrá coste $O(n)$ para la inserción, con n el número de elementos del TAD.	V
El TAD Cola de Prioridad representado por una lista desordenada, tendrá coste $O(l)$ para el borrado.	F

## MONTÍCULO

Al realizar un recorrido en inorden de un montículo obtenemos una sucesión de claves ordenadas.	F
En un montículo doble de altura h se pueden almacenar $2^h - 1$ claves.	F
En el proceso de inserción en un montículo máximo insertaremos en la siguiente posición libre para que siga siendo un árbol binario lleno.	F
Para todos los nodos de un montículo, se cumple que el número de nodos de su hijo izquierdo es mayor o igual que el de su hijo derecho.	F
En un montículo de altura k el número total de nodos es $2^k - 1$ .	F
En un montículo doble todas las claves del montículo máximo son mayores que las del montículo mínimo.	F
Un montículo mínimo es un árbol binario completo, en el que se ha establecido una relación de orden parcial por la que el elemento mínimo del conjunto aparece en el nodo raíz.	V
En un montículo, el número de claves en el hijo izquierda de la raíz es mayor o igual que en su hijo derecha.	V
El montículo o HEAP mínimo es un árbol binario lleno que además es árbol mínimo.	F
El montículo o HEAP mínimo es una estructura tipo árbol en la que se tiene al elemento mínimo del conjunto en el nodo hoja más a la izquierda.	F
Si se implementa el algoritmo de ordenación de un vector "heapsort" utilizando un heap máximo los elementos quedan ordenados en el vector de forma descendente.	F
El siguiente vector representa un montículo máximo: 10 5 3 1 2	V

## ÁRBOL LEFTLIST

En un árbol Leftlist que a su vez es un árbol binario lleno, el camino mínimo de la raíz es igual a la altura del árbol.	V
En un árbol LeftList, el camino mínimo del nodo raíz siempre es mayor que 1	F
Para todo nodo de un árbol Leftlist, se cumple que el número de nodos de su hijo izquierdo es mayor o igual que el de su hijo derecho.	F
En un árbol LeftList, el camino mínimo de cualquier nodo es mayor o igual que cero.	V
Para todo nodo de un árbol Leftlist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.	F

## COMPLEJIDAD

La representación de conjuntos mediante vectores de bits tiene una complejidad espacial proporcional al tamaño del conjunto universal.	V
Dado un árbol 2-3 de altura h con n ítems con todos sus nodos del tipo 2-nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 h)$ .	F
El TAD Cola de Prioridad tendrá el mismo coste para la operación de borrado de un ítem tanto si se representa por una lista ordenada como desordenada.	F
El TAD Cola de Prioridad representado por una lista desordenada, tendrá coste $O(l)$ para el borrado.	F
En la escala de complejidades, la complejidad logarítmica es menor que la lineal.	V
La complejidad temporal de la operación insertar en una lista es independiente de su implementación.	F
La complejidad temporal en su caso mejor de la operación de búsqueda de un Trie con nodos terminales es de $\Omega(l)$ .	V
La complejidad temporal de la operación apilar en una pila siempre es $O(l)$ .	F
La complejidad de las operaciones de equilibrado de los árboles AVL sugiere que deben utilizarse sólo si las inserciones son considerablemente más frecuentes que las búsquedas.	F
La complejidad temporal de las operaciones de inserción y búsqueda en un árbol de búsqueda digital están en función del número de bits de la clave.	V
La complejidad de la intersección de dos conjuntos representados como listas ordenadas de tamaño 'n' es $O(n)$ .	V
La complejidad temporal en el peor caso y en el mejor caso de las operaciones inserción y borrado en un AVL son lineal y logarítmica respecto al número de nodos en el árbol.	F
En la representación de conjuntos mediante vectores de bits, la complejidad espacial es proporcional al tamaño del conjunto representado.	F
En la complejidad espacial de una pila utilizando estructuras dinámicas es constante respecto al tamaño de la misma.	F
El tiempo requerido por un algoritmo expresado en función de la talla del problema se llama complejidad espacial de algoritmo.	F
El coste temporal de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol.	V
La complejidad temporal en el peor caso de insertar un elemento en un vector ordenado o en una lista ordenada es la misma.	V
La complejidad de la Unión de dos conjuntos implementados como listas no ordenadas de tamaño 'n' y 'm' respectivamente es $O(n*m)$ .	V

La complejidad temporal de la diferencia de dos conjuntos de cardinalidad 'n', representados como listas ordenadas, es $O(n^2)$ .	F
El coste temporal en el peor caso de la operación de inserción en un árbol 2-3-4 es $\log_2(n+1) \approx \log_2(n)$ siendo 'n' el número total de ítems.	V
La complejidad temporal de insertar un elemento en un vector ordenado o en una lista ordenada es la misma.	V
La complejidad temporal para la inserción de un elemento en una lista ordenada y en otra no ordenada es siempre lineal con el número de elementos en ambos casos.	F
En los conjuntos representados como listas enlazadas ordenadas, la complejidad temporal de la operación 'pertenencia de un elemento al conjunto' es $O(n)$ , siendo n el número de elementos.	V
El TAD Cola de Prioridad representado por un montículo, tendrá las siguientes complejidades: $O(l)$ para el borrado y $O(\log n)$ para la inserción, siendo n el número de elementos.	F
La complejidad temporal de la diferencia de dos conjuntos de cardinalidad 'n' representados como listas ordenadas es $O(n)$ .	V
La complejidad temporal de obtener un elemento en un vector ordenado o en una lista ordenada es la misma.	F
La operación inserción en un trie tiene complejidad temporal lineal con el número de elementos.	F
En un grafo dirigido con K aristas y N vértices, una complejidad de $O(K)$ es equivalente a la complejidad de $O(N^2)$ .	V
El TAD Cola de Prioridad representado por una lista ordenada, tendrá las siguientes complejidades: $O(l)$ para el borrado y $O(n)$ para la inserción.	V
La complejidad temporal de la operación desfilar utilizando vectores (con un índice que indica la cima) o listas es la misma.	V
La complejidad temporal en el peor caso de la operación de inserción en un árbol 2-3 es $\log_2(n+1)$ .	V
En los conjuntos representados como listas no ordenadas, la complejidad temporal de la operación 'diferencia de conjuntos' es $O(n)$ .	F
El caso peor de la búsqueda es más eficiente en una lista ordenada que en una lista cuyos elementos no están ordenados.	F
La complejidad temporal de la búsqueda en un trie es lineal respecto al número de palabras almacenadas.	F
La operación de insertar un elemento en una lista ordenada tiene el mismo coste que la de insertar un elemento en un vector ordenado.	V

## SEMÁNTICA

La extensión de un TAD se obtiene añadiendo nuevos tipos sobre los ya creados	V
<p>La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C:Conjunto;x:Ítem):</p> <p>Operación(Crear) <math>\iff</math> 1</p> <p>Operación(Insertar(C,x)) <math>\iff</math> 3 + Operación(C)</p>	F
<p>La semántica de la operación base que actúa sobre una pila y devuelve el primer elemento apilado es la siguiente (p:pila; x:ítem)</p> <p>Base(crear())=error_item()</p> <p>Base(apilar(crear(),x))=x</p> <p>Base(apilar(p,x))=base(x)</p>	F
Las operaciones constructoras modificadoras permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.	F
<p>La semántica de la operación obtener en una lista con acceso a posición es la siguiente:</p> <p>Obtener(crear(),p)=error_item()</p> <p>Si p==primera(IC(l,x)) entonces obtener(IC(l,x),p)=x</p> <p>Sino obtener(IC(l,x),p)=IC(obtener(l,p),x)</p>	F
<p>La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas(C:ConjuntoConClavesRepetidas;x,y:ítem)</p> <p>Eliminar(Crear,x)? Crear</p> <p>Eliminar(Insertar(C,x),y)?</p> <p>Si (x==y) entonces Eliminar(C,y) sino Insertar(Eliminar(C,y),x)</p>	V
<p>La semántica de la operación base que actúa sobre una pila y devuelve el primer elemento apilado es la siguiente (p:pila; x:ítem)</p> <p>Base(crear())=error_item()</p> <p>Base(apilar(crear(),x))=x</p> <p>Base(apilar(p,x))=base(p)</p>	V

<p>La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C:ConjuntoConClavesRepetidas;x,y:ítem)</p> <p>Insertar(Insertar(C,x),y) <math>\iff</math></p> <p>Si(x==y) entonces Insertar(C,x) sino Insertar(Insertar(C,y),x)</p>	F
Las operaciones auxiliares de un TAD también se les llama ocultas o privadas.	V
<p>La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas(C:Conjunto;x,y:ítem)</p> <p>Eliminar(Crear,y) <math>\iff</math> Crear</p> <p>Eliminar(Insertar(C,x),y) <math>\iff</math></p> <p>Si (x==y) entonces C sino Insertar(Eliminar(C,y),x)</p>	F
<p>Sea el siguiente TAD: MODULO NATURALEXAMEN</p> <p>TIPO NATURAL</p> <p>OPERACIONES</p> <p>Uno: <math>\rightarrow</math> natural; siguiente: natural<math>\rightarrow</math>natural</p> <p>Sumar: natural natural <math>\rightarrow</math> natural</p> <p>FMODULO</p> <p>Si N es un natural, entonces N= sumar(uno,siguiente(uno)) es un uso sintácticamente incorrecto de la operación sumar.</p>	F
Una operación del TAD X que tenga la sintaxis Crear() $\rightarrow$ X es una operación constructora generadora.	V
Las operaciones generadoras son aquellas que permiten generar todos los valores del TAD a especificar.	V
La operación de lista: Longitud: (LISTA) $\rightarrow$ NATURAL es una operación consultora.	V
<p>Altura: Árbol <math>\rightarrow</math> Natural</p> <p>Si A <math>\in</math> Árbol, B <math>\in</math> Ítem, entonces b=Altura(A) es un uso sintácticamente correcto de la operación.</p>	F
El calificativo "abstracto" asociado a los tipos de datos hace referencia a que lo que importa es cómo se hacen las cosas y no lo que se hace.	F
Los enriquecimientos no forman parte de la definición de TAD.	V
En cualquier tipo de datos lineal cada elemento tiene un único sucesor y varios predecesores.	F
La elección de la estructura de datos que soportará el TAD debe tomarse en la fase de especificación, no en la fase de implementación.	F



En cualquier tipo de datos lineal cada elemento tiene un único sucesor y un único predecesor.	V
Una expresión está en forma reducida si contiene operaciones que pertenecen solo al conjunto de los constructores.	V
El tipo de datos vector se define como un conjunto en el que sus componentes ocupan posiciones consecutivas de memoria.	F
La operación <code>BorrarItem</code> , que borra todas las ocurrencias del ítem $i$ que se encuentren en la lista, tiene la siguiente sintaxis y semántica:  $\text{BorrarItem: LISTA, ITEM} \rightarrow \text{LISTA}$ $\text{BorrarItem( IC(L I ,j),i)} = \text{si } (i=j) \text{ entonces BorrarItem(L I, i)}$ Sino $\text{IC (BorrarItem(L I,i),j)}$	V
Espacia: $\text{PILA} \rightarrow \text{BOOLEAN}$ Si $P$ y $Q$ son pilas: $Q = \text{Espacia}(P)$ , es un uso sintácticamente correcto de la operación.	F
Longitud: $\text{LISTA} \rightarrow \text{NATURAL}$ Si $L$ es una lista, $a$ es un ítem de la lista: $a = \text{Longitud}(L)$ es un uso sintácticamente incorrecto de la operación.	V
Los enriquecimientos forman parte de la definición de un TAD.	F
La operación <code>crear_pila()</code> es constructora modificadora.	F
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas  $(C: \text{ConjuntoConClavesRepetidas}; x, y: \text{Ítem}):$ $\text{Insertar}(\text{Insertar}(C, x), y)$ $\text{Insertar}(\text{Insertar}(C,y), x)$	V
Para definir la semántica de una operación de un tipo de datos sólo se pueden utilizar las operaciones constructoras generadoras.	F
Si no definimos la semántica de un tipo, las ecuaciones: $\text{siguiente}(\text{siguiente}(\text{uno}))$ y $\text{sumar}(\text{uno}, \text{siguiente}(\text{uno}))$ denotan diferentes valores.	V

## SINTAXIS

En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	F
En las pilas las inserciones y borrados se realizan por el mismo extremo	V
En C++, si un objeto se sale del ámbito entonces se invoca automáticamente al destructor de ese objeto	V
Sea el método <i>Primera</i> perteneciente a la clase <code>TLista</code> que devuelve la primera posición de la lista que lo invoca:  <pre> TPosicion TLista::Primera()      class TLista{ { TPosicion p;                    public:....     p.pos = lis;                  private:     return p; }                  TNode * lis;} </pre> En el método <i>Primera</i> , se invoca a la sobrecarga del operador asignación entre objetos del tipo <i>TPosicion</i> .	F
En una cola circular enlazada, el elemento apuntando a fondo es el primero en desencolar	F
En layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	F
Si definimos un método en la parte privada de una clase, éste será accesible desde los métodos de la propia clase y desde todas sus clases derivadas.	F
Sea el método <i>Primera</i> perteneciente a la clase <code>TLista</code> que devuelve la primera posición de la lista que lo invoca:  <pre> TPosicion TLista::Primera()      class TLista{ { TPosicion p;                    public:....     p.pos = lis;                  private:     return p; }                  TNode * lis;} </pre> En el método <i>primera</i> se invoca al constructor y destructor para el objeto <code>Tposicion p</code> .	V

<p>Sea el método <i>Primera</i> perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:</p> <pre> TPosicion TLista::Primera() { TPosicion p;   p.pos = lis;   return p; } class TLista{ public:.... private:   TNode * lis;} </pre> <p>En la instrucción return p del método Primera se invoca al constructor de copia de TPosicion.</p>	V
<p>La sobrecarga del operador corchete siempre tiene que definirse de la siguiente forma para que pueda aparecer en ambos lados de una asignación: Tótem operador [] (int i);</p>	F
<p>Es posible obtener una representación enlazada de una cola utilizando un único puntero que apuntará al fondo de la cola.</p>	V
<p>En C++, una forma correcta de copiar una cadena es la siguiente:</p> <pre> char a[50] = "Tipos Abstractos de Datos"; char *b; b = new char[strlen(a)]; strcpy(b, a); </pre>	F
<p>En herencia privada los métodos de la clase derivada pueden acceder a la parte pública de la clase base.</p>	V