1. Introducción: Introducción a los TAD:

Las operaciones constructoras modificadoras permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.	F
Los enriquecimientos forman parte de la definición de un TAD	F
Los enriquecimientos no forman parte de la definicion de un TAD	V
EsVacia: PILA -> BOOLEAN. Si P y Q son pilas: Q = EsVacia (P), es un uso sintácticamente correcto de la operación	F
Longitud: LISTA -> NATURAL. Si L es una lista, a es un item de la lista: a = Longitud (L) es un uso sintacticamente incorrecto de la operación	V
Longitud: LISTA -> NATURAL. Si L es una lista, a es un item de la lista: a = Longitud (L) es un uso sintacticamente correcto de la operación	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
Una expresión está en forma reducida si contiene operaciones que pertenecen sólo al conjunto de los constructores	V
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
El tiempo requerido por un algoritmo expresado en función de la talla del problema se llama complejidad espacial del algoritmo	F
Altura: Árbol --> Natural. Si A PERTENECE Árbol, b PERTENECE Ítem, entonces b=Altura (A) es un uso sintácticamente correcto de la operación	F
Las operaciones generadoras son aquellas que permiten generar todos los valores del TAD a especificar	V
Una operación del TAD X que tenga la sintaxis Crear() --> X es una operación constructora generadora.	V
Las operaciones ocultas forman parte de la definición de un TAD	F
Sea el siguiente TAD: MÓDULO NATURALEXAMEN. TIPO natural. OPERACIONES. uno: --> natural; siguiente: natural --> natural. sumar: natural natural --> natural. FMÓDULO. Si N es un natural: N = sumar(uno,siguiente(uno)) es un uso sintácticamente incorrecto de la operación sumar.	F
Una operación consultora devuelve un valor del tipo definido.	F
Para el tratamiento de errores en la especificación, se añaden funciones constantes que devuelven un valor del tipo que causa el error.	V

Para definir la semántica de una operación de un tipo de datos sólo se pueden utilizar las operaciones generadoras constructoras	F
Si no definimos la semántica de un tipo, las ecuaciones: siguiente(siguiente(uno)) y sumar(uno,siguiente(uno)) denotan diferentes valores	V
Las operaciones auxiliares de un TAD también se les llama ocultas o privadas	V
En la escala de complejidades, la complejidad logarítmica es menor que la lineal.	V
Dada una especificación de un TAD, sólo existe una implementación posible	F
Las operaciones constructoras modificadoras permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.	F
Las operaciones constructoras modificadoras permiten obtener cualquier valor del tipo.	F

2. Introducción: C++

En layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	F
La sobrecarga del operador corchete tiene que definirse de la siguiente forma para que pueda aparecer a ambos lados de una asignación: Titem operator[] (int i);	F
La sobrecarga del operador corchete tiene que definirse de la siguiente forma para que pueda aparecer a ambos lados de una asignación: Titem& operator[] (int i);	V
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca: TPosicion TLista::Primera( ) { TPosicion p; p.pos = lis; return p; } class TLista { public: ... private: TNodo *lis; } En el método Primera, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.	V
En layering los métodos de la clase derivada pueden acceder a la parte pública de la clase base.	V
En herencia pública, la parte privada de la clase base es accesible desde los métodos de la clase derivada.	F
Para que se ejecute el constructor de copia de la clase base, éste debe ser invocado explícitamente en la fase de inicialización de la clase derivada	V
En herencia privada los métodos de la clase derivada pueden acceder a la parte pública de la clase base.	V

Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca: TPosicion Tlista::Primera( ) { TPosicion p; p.pos = lis; return p; } class Tlista { public: ... private: TNodo *lis; } En el método Primera se invoca al constructor y destructor para el objeto TPosicion p.	V
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
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	V
En layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	F
Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca: TPosicion Tlista::Primera( ) { TPosicion p; p.pos = lis; return p; } class Tlista { public: ... private: TNodo *lis; } En el método Primera, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.	F
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	F

3. Tipos liniales: Vector, Lista, Pila y Cola.

La operación BorrarItem tiene la siguiente sintaxis y semantica: BorrarItem: LISTA, ITEM -> LISTA. BorrarItem(Crear, i) = Crear. BorrarItem(IC(L1,j), i) = si (i == j) entonces L1. sino IC (BorrarItem (L1, i), j). Esta operación borra todas las ocurrencias del item que se encuentra en la lista	F
La operación BorrarItem tiene la siguiente sintaxis y semantica: BorrarItem: LISTA, ITEM -> LISTA. BorrarItem(Crear, i) = Crear. BorrarItem(IC(L1,j), i) = si (i == j) entonces L1. sino IC (BorrarItem (L1, i), j). Esta operación borra la primera ocurrencia del item que se encuentra en la lista	V

La operación BorrarItem, que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semàntica: BorrarItem: LISTA, ITEM -> LISTA. BorrarItem(Crear, i) = Crear. BorrarItem(IC(L1,j), i) = si (i == j) entonces BorrarItem (L1, i). sino IC (BorrarItem (L1, i), j)	V
La complejidad temporal de obtener un elemento en un vector ordenado o en una lista ordenada es la misma	F
La complejidad temporal de la operación desapilar utilizando vectores (con un indice que indica la cima) o listas es la misma	V
La complejidad temporal para la inserción de un elemento en una lista ordenada y en otra no ordenada siempre es lineal con el número de elementos en ambos casos	F
En cualquier tipo de datos lineal cada elemento tiene un único sucesor y un único predecesor	V
En cualquier tipo de datos lineal cada elemento tiene un único sucesor y varios predecesor	F
El tipo de datos vector se define como un conjunto en el que sus componentes ocupan posiciones consecutivas de memoria	F
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 espacial de una pila utilizando estructuras dinámicas es constante respecto al tamaño de la pila	F
La operación de lista: Longitud: (LISTA) --> NATURAL es una operación consultora	V
Es posible obtener una representación enlazada de una cola utilizando un único puntero que apuntará al fondo de la cola.	V
En una lista se establece un orden secuencial a partir de las posiciones que ocupan sus elementos	V
En una cola representada a partir de una lista enlazada simple con un único puntero al principio de la lista (cabeza de la cola), todas las operaciones de la cola (Cabeza, Encolar, Desencolar y EsVacía) se realizan en tiempo constante.	F
UNA LISTA ES UNA SECUENCIA DE CERO O MÁS ELEMENTOS DE CUALQUIER TIPO DE LA FORMA: EP, ESIG(P), ...	F
Una lista de acceso por posiciones es un secuencia de cero o mas elementos que pueden ser de distinto tipo	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
LA COMPLEJIDAD TEMPORAL DE LA OPERACIÓN APILAR EN UNA PILA SIEMPRE ES O(1).	F
LA COMPLEJIDAD TEMPORAL DE LA OPERACIÓN INSERTAR EN UNA LISTA ES INDEPENDIENTE DE SU IMPLEMENTACIÓN.	F
Un vector es un conjunto ordenado de pares <índice, valor>.	V
La semántica de la operación obtener en una lista con acceso por posición es la siguiente (IC= InsertarCabeza(Lista, Ítem), p: posición, l1: lista, x: ítem):. obtener(crear(),p)=error_item(). si p = = primera(IC(l1,x)) entonces obtener(IC(l1,x),p)=x. sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)	F
La semántica de la operación base que actúa sobre una pila y devuelve el primer elemento apilado es la siguiente (p: pila, x: ítem):. base(crear())=error_item(). base(apilar(crear(),x))=x. base(apilar(p,x))=base(p)	V
En una cola circular enlazada, el elemento apuntado por fondo es el primero a desencolar.	F
La semántica de la operación base que actúa sobre una pila y devuelve el primer elemento apilado es la siguiente (p: pila, x: ítem):. base(crear())=error_item(). base(apilar(crear(),x))=x. base(apilar(p,x))=base(x)	F
LAS PILAS TAMBIÉN SE CONOCEN COMO LISTAS LIFO.	V

4. Árboles: Conceptos generales

El grado de un árbol es el grado mínimo de todos los nodos de ese árbol	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 un árbol lleno, es posible reconstruir dicho árbol	V
Dado un único recorrido de un árbol, es posible reconstruir dicho árbol	F
Dado un único recorrido de cualquier árbol, es posible reconstruir dicho árbol	F
Un árbol binario completo con n nodos y altura k es un árbol binario lleno para esa misma altura	F
El grado de un nodo es el número de items asociados a dicho nodo	F
Existe al menos un árbol, que representa los siguientes recorridos: inorden = YXZT, niveles = XTYZ	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
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
La complejidad temporal del recorrido por niveles es la misma que las de los recorridos in-pre-post orden	V
Un camino en un árbol es una secuencia a1, ..., as de árboles tal que para todo . i PERTENECE {1, ..., s-1}, ai+1 es subárbol de ai.	V
El máximo número de nodos en un nivel i-1 de un árbol binario es 2i-2 , i >= 2	V
El mínimo número de nodos que ha de tener un árbol binario de altura 4 para ser equilibrado respecto a la altura es 7	V
Existe un único árbol binario completo que se puede construir a partir del recorrido en postorden	V
En un árbol binario enhebrado, el primer nodo del recorrido en inorden no tiene hebra izquierda	V
Un árbol de Fibonacci siempre está equilibrado respecto a la altura	V
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 ó 2 hebras ó 2 hijos ó 1 hijo y 1 hebra	F
En un árbol binario enhebrado todas las hojas tienen 2 hebras.	F
Si el nodo a borrar en un árbol binario de búsqueda tiene 2 hijos, éste se puede sustituir por el hijo mayor del subárbol derecho.	F
La profundidad de un subárbol es la longitud del único camino desde la raíz a dicho subárbol.	V
El grado de un árbol es el número mínimo de hijos que pueden tener sus subárboles.	F
Un bosque de grado 3 se puede representar como un árbol binario.	V
El máximo número de nodos en un árbol binario de altura k-1 es 2k - 1, k >= 1.	F
En un árbol cada elemento puede tener varios predecesores, pero como máximo un sucesor.	F

5. Árboles de búsqueda

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 subarbol de la derecha	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 subarbol de la derecha	V
A los árboles generales también se les llama árboles multicamino de búsqueda	F
Cuando realizamos un recorrido en preorden en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor	F
Cuando realizamos un recorrido por niveles en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor	F
El menor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja	F
El mayor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja	F
El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda siempre se encuentra en la raíz.	F
El coste temporal de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	V
El coste temporal de insertar una etiqueta en un árbol binario de búsqueda es logarítmica respecto a la altura del árbol	F
A partir del recorrido por niveles de un árbol binario completo se puede obtener el árbol al que representa.	V
Suponiendo que tenemos un árbol binario de búsqueda lleno con n elementos, la búsqueda del elemento número n/2 según la relación de orden se realiza en tiempo logarítmico.	F
Sólo existen tres formas de recorrido en profundidad en un árbol binario: preorden, inorden y postorden	F
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
El número de nodos de un árbol de Fibonacci de altura h es el máximo número de nodos que puede tener un árbol binario de altura h para que sea AVL.	F

6. Árboles AVL

Un árbol de Fibonacci siempre está balanceado respecto a la altura	V
Un árbol de Fibonacci es un árbol balanceado con respecto a la altura	V
Un árbol de Fibonacci es un árbol completamente equilibrado	F
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho solo se va a efectuar una rotación	V
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 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
Las rotaciones que hay que realizar en los árboles AVL para mantenerlos balanceados tienen un coste temporal lineal con respecto al número de items del árbol	F
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
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
Un árbol completo siempre está balanceado respecto a la altura	V
La complejidad de las operaciones de equilibrado sugiere que estos árboles deben utilizarse sólo si las inserciones son considerablemente más frecuentes que las búsquedas.	F
Un árbol AVL es un árbol binario de búsqueda en el que la diferencia de nodos entre el subárbol izquierdo y derecho es como máximo uno.	F
En un árbol AVL siempre que se inserte una etiqueta hay que realizar una rotación	F
Un árbol completo es un árbol completamente equilibrado	F
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
El borrado en 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
El número mínimo de nodos que tiene un árbol AVL de altura 5 es 12.	V

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.	V
Un árbol AVL es un árbol balanceado respecto al número de nodos de los subárboles.	F

7. Árboles 2-3

Un árbol 2-3 es un árbol 2-camino de búsqueda	F
El número mínimo de elementos que se pueden almacenar en un árbol 2-3 de altura h es 3h-1	F
El número mínimo de elementos que se pueden almacenar en un árbol 2-3 de altura h es 2h-1	V
El mínimo número 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 coste temporal en el peor caso de la operación de inserción en un árbol 2-3-4 es log2(n+1) <=> log2(n) siendo “n” el número total de items	V
Dado un árbol 2-3 de altura h con n items: 2h-1 =< n =< 3h-1	V
Dado un árbol 2-3 con n items con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es O(log2 n)	V
El grado del árbol 2-3 es 2	F
Los nodos de grado 0 de un árbol 2-3 pueden estar en distintos niveles del árbol	F
Sea un árbol 2-3 de altura h, el número total de nodos del árbol está entre 2h-1 y 3h-1	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 completo de altura h	F
Existe un único árbol 2-3 de altura 3 que representa a las etiquetas del 1 al 9.	F
En un árbol 2-3 la altura del árbol sólo aumenta cuando todas las hojas del árbol son de grado tres.	F
Dado un árbol 2-3 con n items con todos sus nodos del tipo 3-Nodo: la complejidad de la operación de búsqueda de un ítem es O(log3 (n+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 items se cumple que: 3h-1 < n < 2h-1	F
La operación de Rotación en un árbol 2-3 se da cuando el hermano del nodo donde se efectúa es del tipo 3-Nodo.	V
Dado un árbol 2-3 de altura h con n items con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es O(log2 h)	F
Dado un árbol 2-3 de altura h con n items con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es O(log3 n)	F

8. Árboles 2-3-4

El árbol 2-3-4 no vacío tiene como mínimo dos claves en cada nodo	F
El árbol 2-3-4 no vacío tiene como mínimo una clave en cada nodo	V
En un árbol 2-3-4 de altura h, el máximo número de etiquetas se da cuando todos los nodos son de tipo 2-nodo.	V
a complejidad temporal en el peor caso de la operación inserción en un árbol 2-3-4 es log2(n+1)	V
La operación de inserción en un árbol 2-3-4 requiere una restructuración del árbol en el camino de vuelta 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 Rojo-Negro es una representación sobre árbol binario de un árbol 2-3-4	V
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3	F
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
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	V
En un árbol 2-3-4 las inserciones siempre se realizan en las hojas.	V
La operación de borrado en un árbol 2-3-4 no requiere una restructuración del árbol en el camino de vuelta de las hojas a la raíz.	V
En un árbol 2-3-4 con n elementos, la altura de dicho árbol se encuentra entre log4(n+1) y log2(n+1).	V

Un árbol 2-3-4 es un árbol 4-camino de búsqueda	V
El número de elementos que hay en un árbol 2-3-4 de altura h está comprendido entre 2h – 1 y 4h – 1	V
En el borrado del árbol 2-3-4 sólo se reduce la altura del árbol cuando p, q y r son 2-nodo.	V
En la inserción de un árbol 2-3-4 sólo crece la altura del árbol cuando se realiza una operación de DIVIDERAIZ.	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 q sea 2-nodo hay que hacer una COMBINACIÓN o una ROTACIÓN.	V
En un árbol 2-3-4 los nodos pueden tener 1, 2, 3 ó 4 hijos.	F

9. Árboles Rojos-Negros

El recorrrido en inorden en un árbol rojo-negro es el mismo que el realizado sobre un árbol binario	V
El recorrrido en preorden en un árbol rojo-negro es el mismo que el realizado sobre un árbol binario	V
El número de hijos negros en cualquier camino desde la raíz a las hojas es siempre el mismo en un árbol rojo-negro	V
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 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 m-camino de búsqueda con m=2	V
Un árbol rojo-negro es un árbol binario balanceado respecto a la altura	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

Las hojas en un árbol rojo-negro están enlazadas a sus padres siempre en color rojo.	F
En un árbol rojo-negro ningún camino desde la raíz a las hojas tiene dos o más hijos negros consecutivos.	F
Un árbol rojo-negro es un árbol binario de búsqueda que representa a un árbol 2-3-4.	V
Todas las operaciones (inserción y búsqueda) de un árbol rojo-negro se realizan en un tiempo O(log2 n), siendo n el número de items.	V
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
Para cualquier nodo de un árbol rojo-negro se cumple que el número de nodos de su subárbol izquierdo es el mismo que el de su subárbol derecho	F

10. Árboles B

El árbol 2-3 es un árbol B m-camino de búsqueda con m=3	V
El árbol 2-3 es un árbol B m-camino de búsqueda con m=2	F
La raíz del árbol B m-camino de búsqueda siempre tiene al menos m/2 claves o etiquetas	F
La raíz del árbol B m-camino de búsqueda no vacío siempre tiene al menos una etiqueta	V
El árbol B m-camino de búsqueda si tiene un nodo que no sea hoja con “x” claves, entonces ese nodo tendrá “x+1” hijos no vacíos.	V
El árbol B m-camino de búsqueda lleno con altura “k” tiene 2k-1 claves.	F
El nodo de un árbol B m-camino de búsqueda puede tener como máximo “m” ítems.	F
l nodo de un árbol B m-camino de búsqueda puede tener como máximo “m” hijos no vacíos.	V
En el árbol B m-camino de búsqueda, si un nodo tiene “x” claves, entonces “m” es menor que “x”	F
El 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 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 siempre se puede transformar en un árbol binario de búsqueda completo	V
Un árbol B siempre se puede transformar en un árbol binario de búsqueda lleno	F
El nodo de un árbol B m-camino de búsqueda con m=4 puede tener como máximo 4 hijos no vacíos.	V
Las operaciones de inserción y borrado de un árbol 2-3 son las mismas que las de un árbol B de grado 3	V
El nodo de un árbol B m-camino de búsqueda con m=100 puede tener como máximo 99 claves.	V
El árbol B m-camino de búsqueda con altura “k” tiene como máximo mk-1 claves.	F
El árbol B m-camino de búsqueda con altura “k” tiene como máximo mk -1claves	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 claves o etiquetas.	V
El grado de un árbol B m-camino de búsqueda es “m”	V
El árbol B m-camino de búsqueda tiene como máximo 2m -1claves.	F
En un árbol B m-camino de búsqueda con m=16, en cualquier nodo excepto la raíz hay 7 ítems como mínimo.	V
En el árbol B, la inserción de una clave sólo se puede realizar en las hojas del árbol.	V
El grado de un árbol B m-camino de búsqueda es “m-1”.	F

11. Conjuntos I (Complejidad espacial/temporal – Algoritmos)

En los conjuntos representados como listas no ordenadas, la complejidad temporal de la operación “diferencia de conjuntos” es O(n).	F
En los conjuntos representados como listas ordenadas, la complejidad temporal de la operación “diferencia de conjuntos” de cardinalidad “n” es O(n).	V
La complejidad temporal de la diferencia de dos conjuntos de cardinalidad “n”, representados como listas ordenadas, es O(n).	V
La complejidad temporal de la diferencia de dos conjuntos de cardinalidad “n”, representados como listas ordenadas, es O(n2).	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
La mejor representación de los conjuntos siempre es el vector de bits porque es la más eficiente espacialmente.	F
En la representación de conjuntos mediante listas el espacio es proporcional al tamaño del conjunto representado.	V
En la representación de conjuntos mediante las listas el espacio es proporcional al tamaño del conjunto universal.	F
La complejidad de la unión de dos conjuntos implementados como listas no ordenadas de tamaño “n” y “m” respectivamente es O(n*m).	V
En la representación de conjuntos mediante los vectores de bits, la complejidad espacial es proporcional al tamaño del conjunto representado.	F
La complejidad de la intersección de dos conjuntos representados como listas ordenadas de tamaño “n” es O(n).	V
El TAD Diccionario es un subtipo del TAD Conjunto	V
La representación de conjuntos mediante vectores de bits tiene una complejidad espacial proporcional al tamaño del conjunto universal.	V
La complejidad temporal de la búsqueda de un elemento en un conjunto de cardinalidad “n”, representado como una lista, es O(n).	V
La representación de conjuntos mediante vectores de bits tiene una complejidad espacial proporcional al tamaño del conjunto universal.	V
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):. Eliminar(Crear, x) <=> Crear. Eliminar(Insertar(C, x), y) <=> . si (x == y) entonces C sino Insertar(Eliminar(C, y), x)	F
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: Ítem):. Insertar(Insertar(C, x), y) <=> . si (x == y) entonces Insertar(C,x) sino Insertar(Insertar(C,y), x)	F
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):. Eliminar(Crear, x) <=> Crear. Eliminar(Insertar(C, x), y) <=> . si (x == y) entonces Eliminar(C, y) sino Insertar(Eliminar(C, y), x)	V
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto (C: Conjunto; x: Ítem):. Operación(Crear) <=> 0. Operación (Insertar(C, x)) <=> 2 + Operación(C)	F

12. Conjuntos II (TAD Diccionario – Dispersión cerrada/abierta)

En el TAD Diccionario con dispersión cerrada, los elementos se almacenan en una tabla de tamaño fijo.	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, con función de redispersión "hi(x)=(H(x) + C*i) 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
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
El proceso de dispersión es opuesto al de ordenación	V
Sea una tabla de dispersión cerrada con estrategia de redispersión hi(x)=(H(x) + C*i) 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
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 una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: hi(14)=(28 + 7*i) MOD 2000, se recorrerán todas las posiciones de la tabla buscando una posición libre.	V
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
El factor de carga de la dispersión cerrada siempre está entre 0 y 1	V
El factor de carga de la dispersión abierta siempre está entre 0 y 1	F
Sea una tabla de dispersión cerrada con función de dispersión H(x)=x 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
Cuando implementamos un TAD Tabla de dispersión cerrada se usa una función de dispersión H tal que H(x) devolverá un valor comprendido desde 0 hasta B, siendo B el número finito de clases en las que dividimos el conjunto.	F
En la dispersión cerrada sólo se producen colisiones entre claves sinónimas	F
Cuando utilizamos una tabla de dispersión cerrada de tamaño B, el número de elementos del conjunto está limitado a B-1 elementos	F
La dispersión cerrada con estrategia de redispersión aleatoria tiene la siguiente función de redispersión: hi(x)=(hi-1(x) + C) MOD B.	V

En la dispersión abierta sólo se producen colisiones entre claves sinónimas	V
Sea la dispersión cerrada con la siguiente función de redispersión: hi(x)=(hi-1(x) + C) MOD B. Dos claves sinónimas (x, y) tendrán la misma secuencia de intentos, es decir, hi(x) = hi(y)	V
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: hi(14)=(28 + 2*i) MOD 2000, se recorrerán todas las posiciones de la tabla buscando una posición libre.	F
En la busqueda de elementos en una tabla de dispersion cerrada, hay que distinguir durante la búsqueda las casillas vacías de las suprimidas.	V
En el TAD Diccionario con dispersión cerrada, con función de redispersión "hi(x)=(H(x) + k(x)*i) 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 "hi(x)=(H(x) + k(x)*i) MOD B", "B=10" y "k(x)=(x MOD B-1)+1", para la clave x=2 se recorrerán todas las posiciones de la tabla buscando una posición libre.	F

13. Conjuntos IV (TAD Cola de prioridades)

El TAD Cola de Prioridad representado por una lista ordenada, tendrá las siguientes complejidades: O(1) para el borrado, y O(n) para la inserción, siendo n el número de elementos.	V
El TAD Cola de Prioridad representado por un montículo, tendrá las siguientes complejidades: O(1) para el borrado, y O(log n) para la inserción, siendo n el número de elementos	F
El montículo o HEAP mínimo es un árbol binario lleno que además es árbol mínimo.	F
El montículo o HEAP mínimo es un árbol binario completo que además es árbol mínimo.	V
El TAD Cola de Prioridad representado por una lista desordenada, tendrá coste O(1) para el borrado.	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
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 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 de altura k el número total de nodos es 2k-1	F

En un montículo doble todas las claves del montículo máximo son mayores que las del montículo mínimo	F
Para todos los nodos de un montículo, se cumple que el número de nodos de su subárbol izquierdo es menor que el número de nodos de su subárbol derecho	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
Un montículo mínimo es un árbol binario lleno que además es árbol mínimo	F
En un HEAP MAXIMO los elementos de las hojas son los que tienen mayores valores en sus claves	F
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 2h-1 claves.	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

14. Conjuntos V (Árboles leftist)

Para todo nodo de un árbol Leftist, se cumple que el número de nodos de su hijo izquierdo es menor que el de su hijo derecho.	F
Para todo nodo de un árbol Leftist, 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 binario, el camino mínimo de la raíz es igual a la altura del árbol.	F
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es mayor o igual que la de su hijo derecho.	F
En un árbol binario lleno, el camino mínimo de la raíz (longitud del camino más corto hasta un árbol vacío) es igual a la altura del árbol.	V
En un árbol binario, el camino mínimo de la raíz (longitud del camino más corto hasta un árbol vacío) es igual a la altura del árbol.	F
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.	F
Un árbol 2-3-4 cumple las condiciones para ser también un árbol Leftist.	F
Un árbol binario completo cumple las condiciones para ser también un árbol Leftist.	V

Un árbol binario lleno cumple las condiciones para ser también un árbol Leftist.	V
Todo árbol binario de búsqueda lleno es un árbol Leftist mínimo.	F
En un árbol Leftist 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 binario completo, el camino mínimo de la raíz es igual a la altura del árbol.	F
En un árbol Leftist, el camino mínimo de los nodos hoja siempre es 1.	V

15. Conjuntos VI (Los tries y los árboles digitales)

Las operaciones de búsqueda e inserción de un elemento en un trie tienen complejidad temporal lineal con el número de elementos que pertenecen al conjunto.	F
La representación del nodo de un trie utilizando un vector de punteros siempre es más eficiente espacialmente que utilizar una lista de punteros.	F
La representación del nodo de un trie utilizando un vector de punteros siempre es más eficiente temporalmente que utilizar una lista de punteros.	V
La altura de un trie en su peor caso vendrá determinada por la cadena de mayor longitud almacenada en el árbol.	V
La altura de un trie en su peor caso vendrá determinada por la cadena de menor longitud almacenada en el árbol.	F
En un trie, la elección de la siguiente rama a explorar en un proceso de búsqueda, viene determinada por la siguiente letra de la palabra a buscar.	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
La altura máxima de un árbol de búsqueda digital es “n+1”, siendo n el número de bits de la clave.	V
La complejidad temporal de las operaciones de inserción y búsqueda en un árbol de búsqueda digital están en función del numero de bits de la clave.	V
El proceso de búsqueda en un árbol de búsqueda digital es igual que el del árbol binario de búsqueda.	F
La complejidad temporal de las operaciones de inserción y búsqueda en un árbol de búsqueda digital es O(n) siendo n el numero de elementos del conjunto.	F
La complejidad temporal en su caso mejor de la operación de búsqueda en un trie con nodos terminales es de CASOMEJOR(1)	V

La altura de un trie con nodos terminales será como mínimo la longitud de la cadena más larga almacenada.	F

16. Grafos

En un multigrafo pueden existir infinitas aristas para un numero “n” de vértices.	V
En un grafo dirigido pueden existir infinitas aristas para un numero “n” de vértices.	F
En un grafo dirigido con K aristas y N vértices, una complejidad de O(K) es equivalente a la complejidad de O(N2).	V
Un grafo no dirigido de n vértices es un árbol libre si está libre de ciclos y tiene “n-1” aristas	V
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
Ciclo es cualquier camino en el que el vértice primero y último coinciden.	F
En un grafo dirigido pueden existir infinitas aristas para un numero “n” de vértices.	F
Dado un grafo dirigido, siempre se cumple que Adyacencia_de_Salida(x) INTERSECCION Adyacencia_de_Entrada(x) = CONJUNTO_VACIO, donde x es un vértice del grafo.	F
En un grafo no dirigido de “n” vértices pueden existir infinitas aristas.	F
Sea G=(V,A) un grafo dirigido. Diremos que G”=(V”,A”) es un árbol extendido de G <=> V”=V, A” INCLUIDO A, PARATODO v PERTENECE V” ==> gradoE(v) =< 1	V
Al representar un grafo de N vértices y K aristas con una matriz de adyacencia, la operación de calcular la adyacencia de salida de un vértice, tiene una complejidad de O(N).	V
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
Al representar un grafo dirigido de N vértices y K aristas con una matriz de adyacencia, la matriz será simétrica respecto la diagonal principal.	F
Los arcos de retroceso de un recorrido en profundidad de un grafo dirigido, nos indican la presencia de un ciclo.	V
Al representar un grafo de N vértices y K aristas con una lista de adyacencia, la operación de hallar la adyacencia de entrada de un vértice, tiene una complejidad de O(K).	V

La siguiente secuencia de nodos de 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
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de avance y de cruce es un grafo acíclico dirigido.	V
Los árboles extendidos de un grafo tienen que ser necesariamente un árbol binario.	F
Sea G un grafo no dirigido de n vértices. Si G tiene “n-1” aristas, entonces nunca podría tener un ciclo.	F
Al clasificar las aristas de un grafo no dirigido en un recorrido en profundidad, sólo existen aristas de árbol y de retroceso.	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 del árbol”.	F
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de retroceso es un grafo acíclico dirigido.	F