

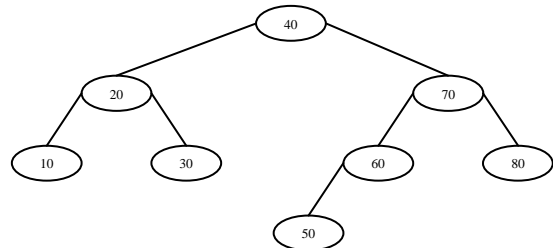
Examen TAD/PED diciembre 2003

- Normas:**
- Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **Apellidos, Nombre**. Cada pregunta se escribirá en folios diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación de notas de exámenes:** 9 de diciembre. **ÚNICA fecha de revisión de exámenes:** 17 de diciembre. El lugar y la hora se publicará en el campus virtual.
- Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Sea un vector de números naturales. Utilizando exclusivamente las operaciones *asignar* y *crear*, define la sintaxis y la semántica de la operación *eliminar* que borra las posiciones pares del vector marcándolas con "-1". Para calcular el resto de una división, se puede utilizar la operación MOD.

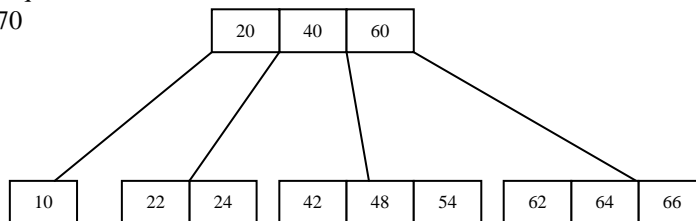
2. a) Dado el siguiente árbol AVL, realizar las siguientes operaciones en el siguiente orden, indicando en cada caso las transformaciones realizadas, y empleando los siguientes criterios: en caso de un nodo interior, sustituir por el mayor de la izquierda.

- Insertar las etiquetas 45, 85, 110, 5, 1
- Borrar las etiquetas: 60, 40



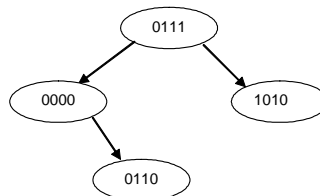
b) Dado el siguiente árbol 2-3-4, realizar las siguientes operaciones en el siguiente orden, indicando en cada caso las transformaciones realizadas, y empleando los siguientes criterios: tomar *r* como el hermano de la derecha, y en caso de un nodo interior, sustituir por el mayor de la izquierda.

- Insertar las etiquetas 50, 26, 56, 70
- Borrar las etiquetas: 22, 48, 24



NOTA: Si el árbol que se obtiene en cualquiera de los apartados no es el correcto, no se corregirán los siguientes apartados.

3. Insertar los ítems que vienen a continuación en el siguiente conjunto que está representado como un árbol digital: 1100, 1101, 1011, 1111, 1110, 0111, 0100.

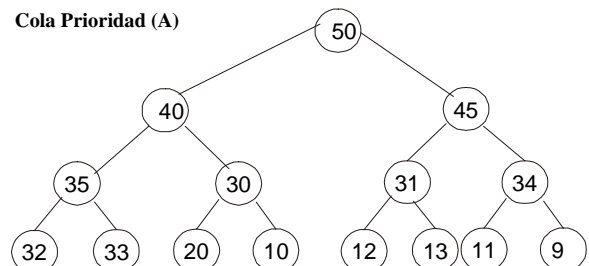


4. a) Sobre la cola de prioridad máxima A representada como un árbol leftist, realizar una operación de borrado.

b) Dada la clase *ArbolLFT*, escribir el código en C++ de la función *void ArbolLFT::cMIN(void)* que recalcula (a partir de los caminos mínimos ya almacenados en sus descendientes) y almacena (en *cmin*) el camino mínimo del nodo al que apunta *p*. Escribir una explicación del algoritmo elegido con una longitud máxima de 3 líneas. (NOTA: Los errores de compilación se puntuarán de forma negativa).

<pre> class ArbolLFT { ... void cMIN(void); ... private: TNode* p; }; </pre>	<pre> class TNode { friend class ArbolLFT; ... private: int clave; int cmin; ArbolLFT izq, der; }; </pre>
--	---

Cola Prioridad (A)



Examen TAD/PED diciembre 2003. Soluciones

1)

eliminar: vector \rightarrow vector

Var v:vector; i,x:natural;

eliminar(crear()) = crear()

si (i MOD 2) == 0

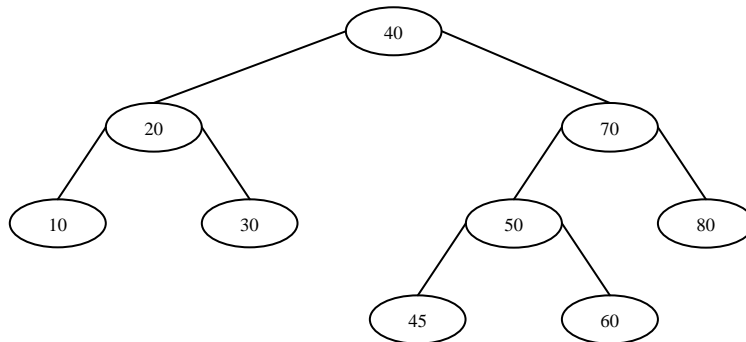
entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,-1)

sino eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x)

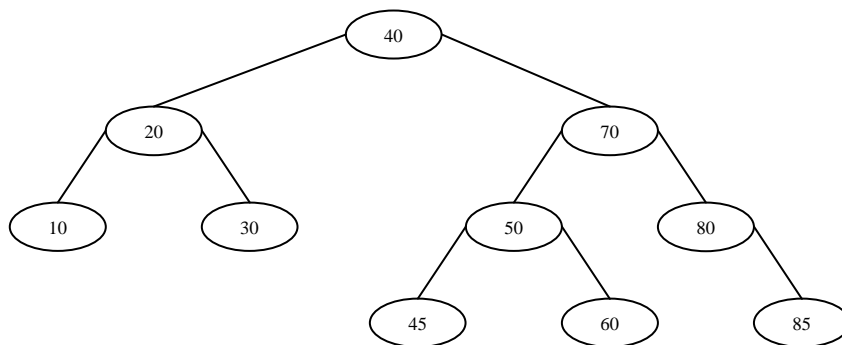
2)

a)

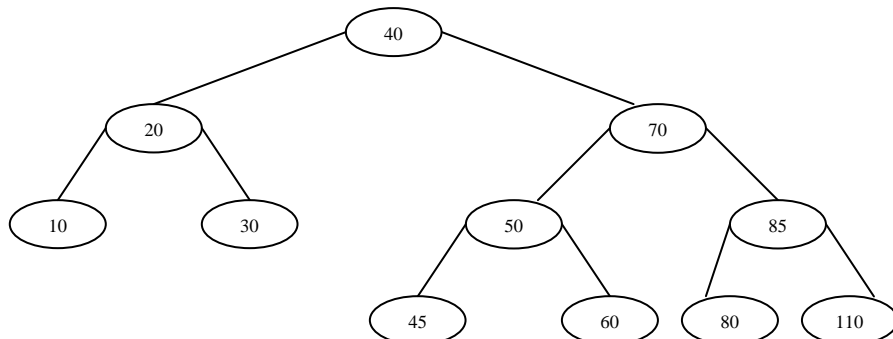
Insertar 45: ROTACIÓN II



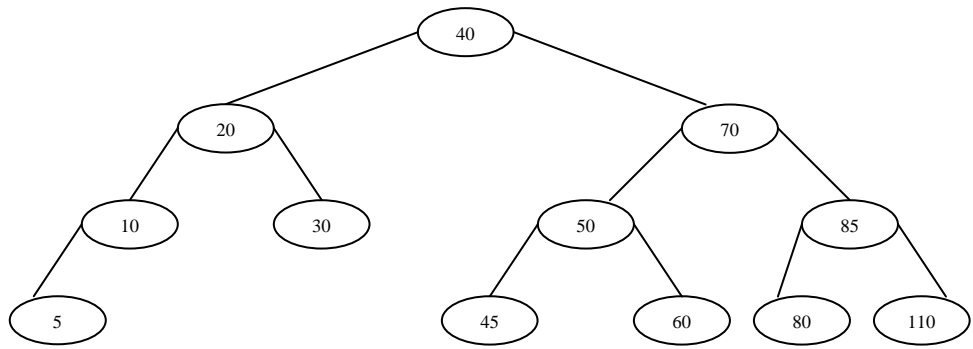
Insertar 85:



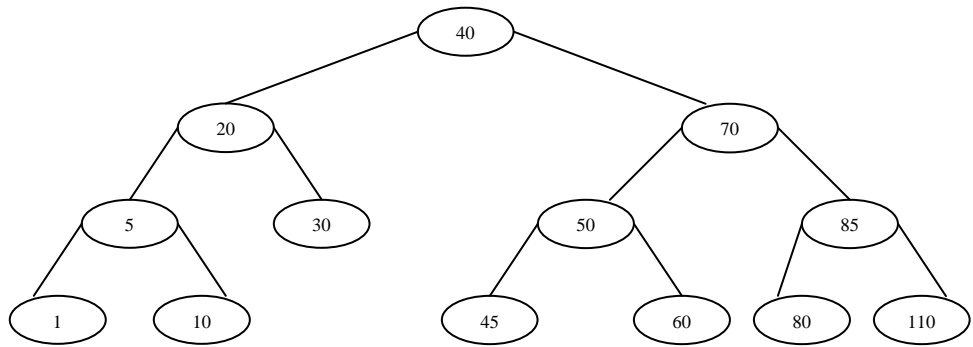
Insertar 110: ROTACIÓN DD



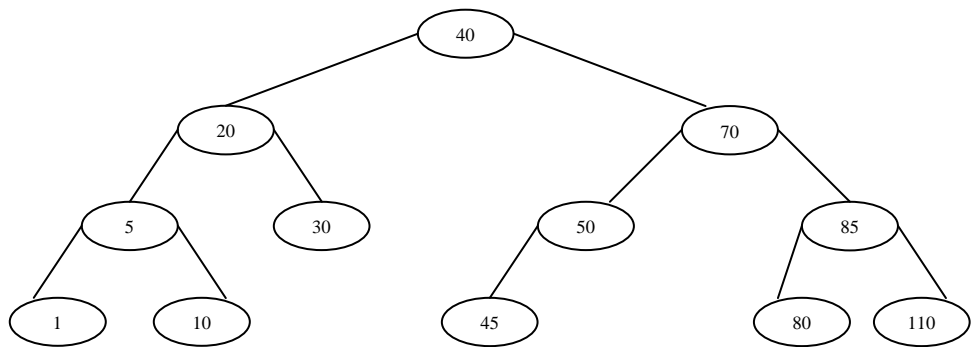
Insertar 5:



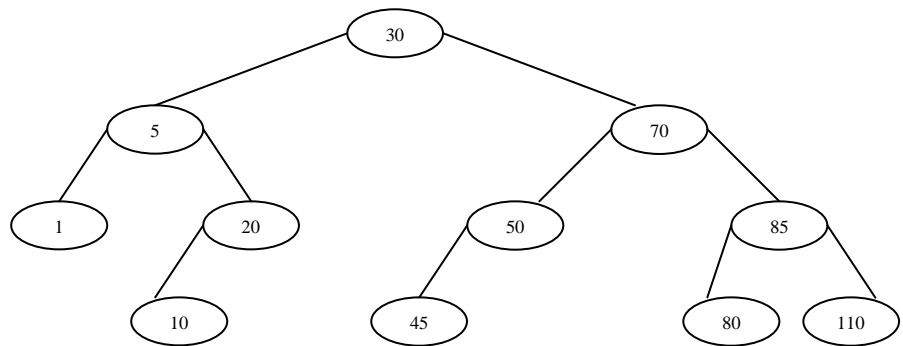
Insertar 1: ROTACIÓN II



PUNTO 2
Borrar 60:

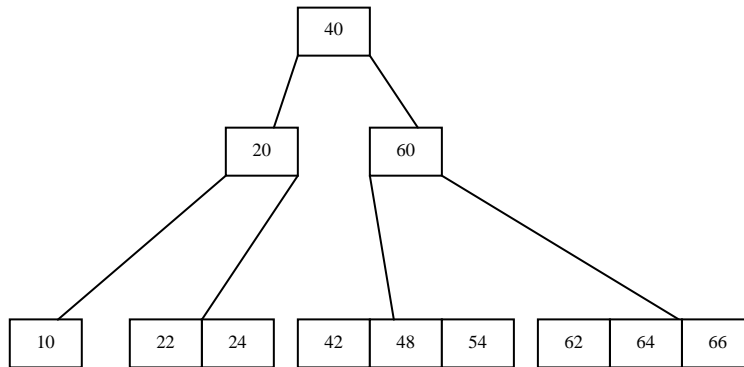


Borrar 40: ROTACIÓN II

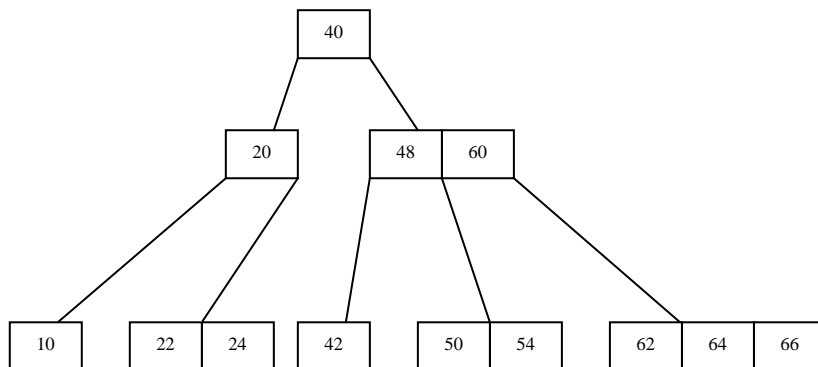


b)

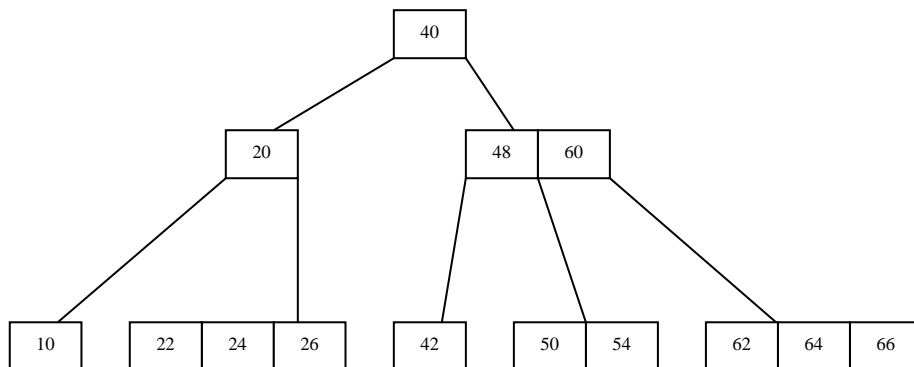
Insertar 50: DIVIDERAIZ



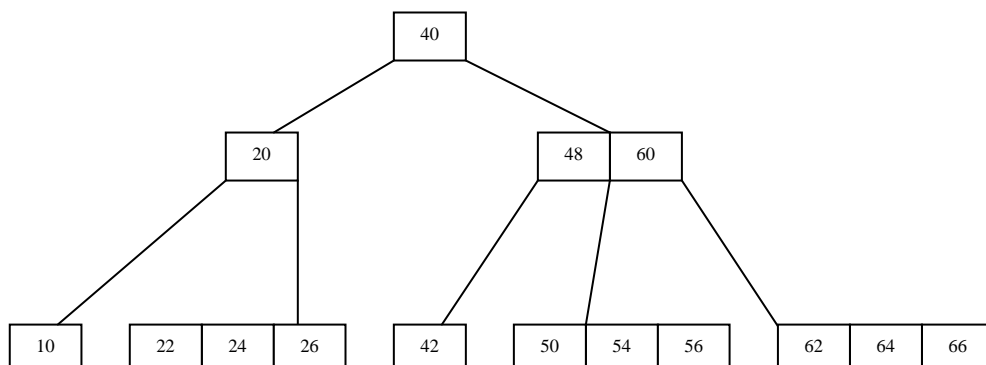
DIVIDEHIJODE2



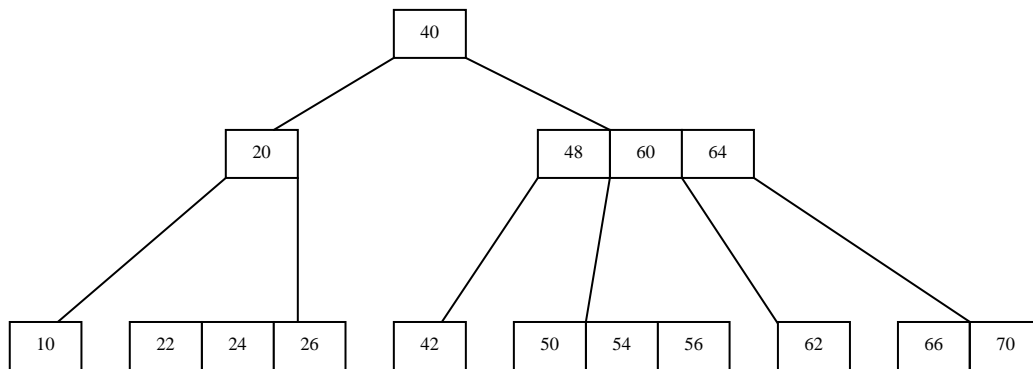
Insertar 26:



Insertar 56:

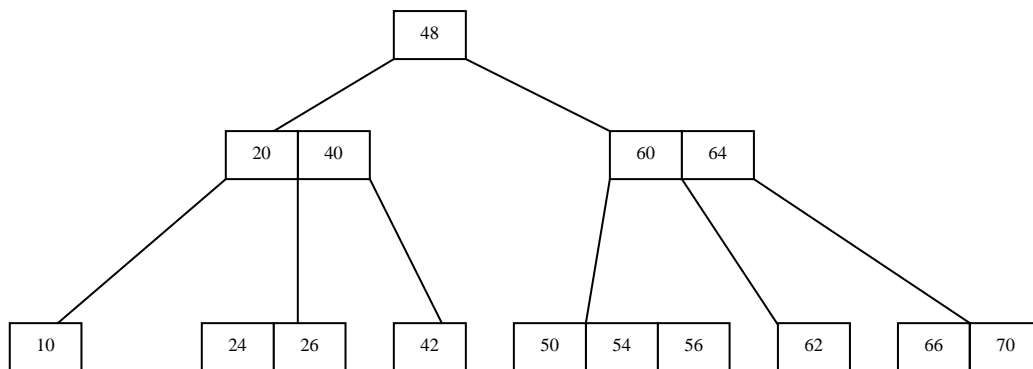


Insertar 70: DIVIDEHIJODE3

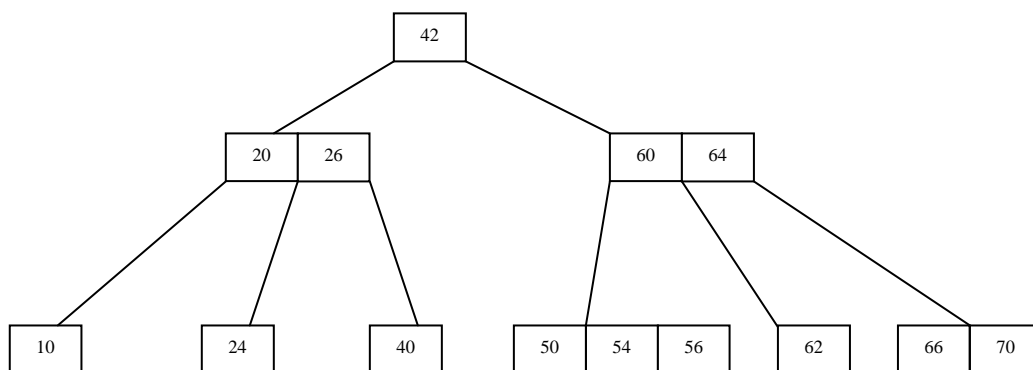


PUNTO 2

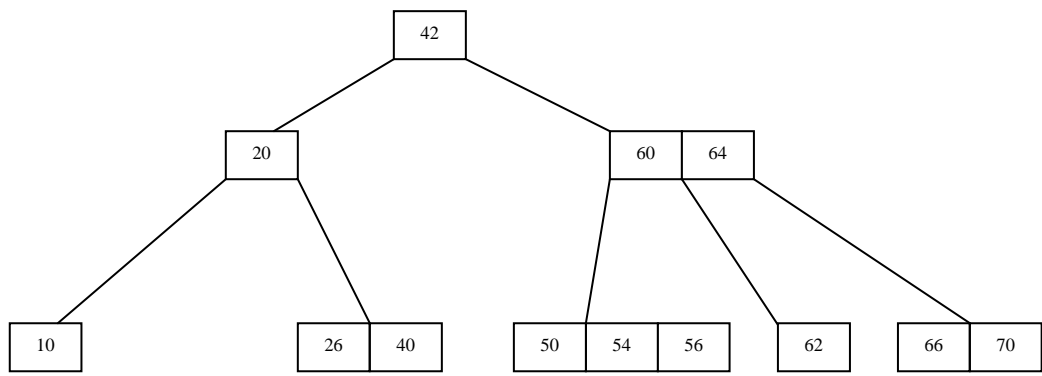
Borrar 22: q es 2-nodo y r es 3-nodo (ROTACIÓN)



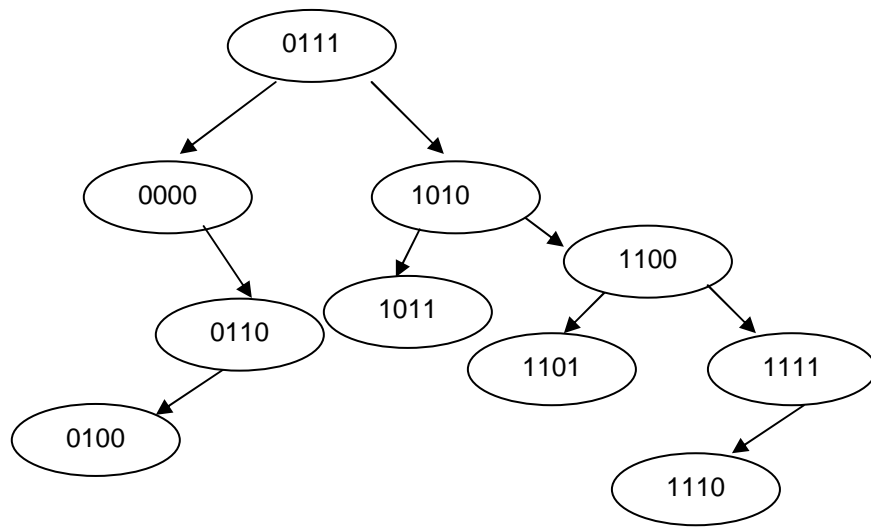
Borrar 48: q es 2-nodo y r es 3-nodo (ROTACIÓN)



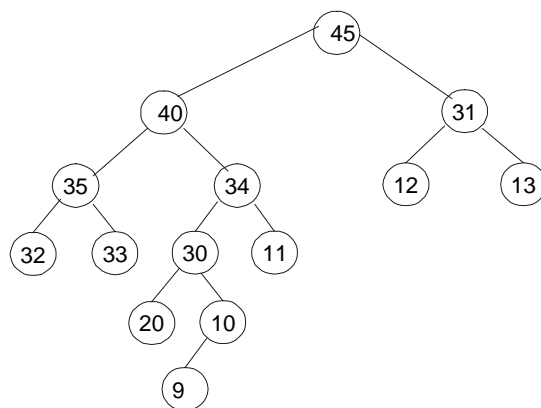
Borrar 24: q es 2-nodo y r es 2-nodo (COMBINACIÓN)



3) Resultado:



4) a)



b)

```
void
ArbolLFT::cMIN(void) {
    if (p != NULL) {
        if (p->der.p == NULL)
            p->cmin = 1;
        else
            p->cmin = p->der.p->cmin + 1;
    }
}
```

Apellidos:

Nombre:

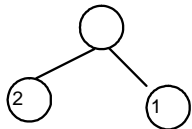
Convocatoria:

DNI:

Examen TAD/PED diciembre 2003

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - **El test vale un 40% de la nota de teoría.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	V 1.
En C++, la parte privada de una clase sólo es accesible por los métodos de la propia clase.	<input type="checkbox"/>	<input type="checkbox"/>	F 2.
El tipo posición en una lista con acceso por posición se puede instanciar a diferentes tipos de objetos.	<input type="checkbox"/>	<input type="checkbox"/>	V 3.
El recorrido en postorden es el inverso especular del recorrido en preorden para un árbol binario dado.	<input type="checkbox"/>	<input type="checkbox"/>	V 4.
En un árbol AVL las inserciones siempre se realizan en las hojas.	<input type="checkbox"/>	<input type="checkbox"/>	V 5.
Dado un árbol 2-3 de altura h con n items con todos sus nodos del tipo 3-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_3 h)$.	<input type="checkbox"/>	<input type="checkbox"/>	F 6.
En un árbol 2-3-4 las reestructuraciones se realizan desde la raíz hacia las hojas.	<input type="checkbox"/>	<input type="checkbox"/>	V 7.
En un árbol rojo-negro, el número de enlaces negros ha de ser mayor que el de enlaces rojos.	<input type="checkbox"/>	<input type="checkbox"/>	F 8.
La altura del árbol B m-camino de búsqueda es " $\log_m n$ ", con " n =número total de claves".	<input type="checkbox"/>	<input type="checkbox"/>	F 9.
La función de redispersión en una tabla hash abierta, para que se recorran todas las posiciones del vector, tiene que cumplir que el valor de B sea primo.	<input type="checkbox"/>	<input type="checkbox"/>	F 10.
El siguiente árbol es un montículo doble: 	<input type="checkbox"/>	<input type="checkbox"/>	F 11.
Todo árbol Leftist cumple las condiciones para ser un árbol binario de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	F 12.
Al representar un grafo dirigido 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 $O(N^2)$.	<input type="checkbox"/>	<input type="checkbox"/>	V 13.
En C++, el constructor de copia sustituye al operador asignación.	<input type="checkbox"/>	<input type="checkbox"/>	F 14.

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen TAD/PED diciembre 2004

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - **El test vale un 40% de la nota de teoría.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En un Tipo Abstracto de Datos la manipulación de los objetos o valores de un tipo, sólo depende del comportamiento descrito en su especificación y es independiente de su implementación.	<input type="checkbox"/>	<input type="checkbox"/>	V 1.
La complejidad temporal (en su peor caso) de la operación de insertar un elemento en una cola circular enlazada que no admite elementos repetidos es $O(n)$.	<input type="checkbox"/>	<input type="checkbox"/>	V 2.
La complejidad temporal de la operación <i>apilar</i> en una pila es independiente de su implementación.	<input type="checkbox"/>	<input type="checkbox"/>	F 3.
Sea el TIPO <i>arbin</i> definido en clase. La semántica de la operación <i>nodos</i> es la siguiente: <i>var i,d:arbin; x:item;</i> <i>nodos(crear_arbin())=0</i> <i>nodos(enraizar(i,x,d))=nodos(i)+nodos(d)</i>	<input type="checkbox"/>	<input type="checkbox"/>	F 4.
En un árbol AVL, el factor de equilibrio de un nodo se define como el valor absoluto de la altura del subárbol de la derecha menos la altura del subárbol de la izquierda.	<input type="checkbox"/>	<input type="checkbox"/>	F 5.
El mínimo número de elementos que se puede almacenar en un árbol 2-3 de altura <i>h</i> coincide con el número de elementos que hay en un árbol binario lleno de altura <i>h</i> .	<input type="checkbox"/>	<input type="checkbox"/>	V 6.
Al insertar un único ítem en un árbol 2-3-4 puede ocurrir que haya que realizar dos operaciones: DIVIDERAIZ (p) y DIVIDEHIJODE2 (p, q).	<input type="checkbox"/>	<input type="checkbox"/>	V 7.
La inserción de una etiqueta en un árbol rojo-negro se efectuará creando un hijo de color rojo.	<input type="checkbox"/>	<input type="checkbox"/>	V 8.
Un árbol binario de búsqueda con altura 7 y 127 nodos es un árbol B con $m=3$	<input type="checkbox"/>	<input type="checkbox"/>	V 9.
En la operación de búsqueda de un elemento en un conjunto ordenado representado mediante un vector de elementos ordenado (representación no enlazada de una lista), se puede utilizar el algoritmo de búsqueda binaria.	<input type="checkbox"/>	<input type="checkbox"/>	V 10.
El factor de carga de la dispersión abierta siempre está entre 0 y 1.	<input type="checkbox"/>	<input type="checkbox"/>	F 11.
En un montículo doble, un elemento “i” del montículo mínimo tiene como máximo un elemento simétrico “j” del montículo máximo.	<input type="checkbox"/>	<input type="checkbox"/>	V 12.
Un árbol binario de búsqueda cumple las propiedades de un árbol Leftist.	<input type="checkbox"/>	<input type="checkbox"/>	F 13.
La siguiente secuencia de nodos de un grafo es un ciclo: 1,2,3,1.	<input type="checkbox"/>	<input type="checkbox"/>	V 14.

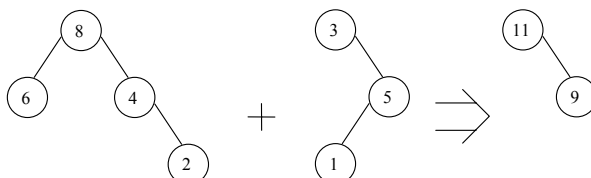
Examen TAD/PED diciembre 2004

- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **Apellidos, Nombre**. Cada pregunta se escribirá en folios diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor**. Este examen vale el 60% de la nota de teoría.
 - **Publicación de notas de exámenes:** 13 de diciembre. **Fecha de revisión de exámenes:** El lugar, fecha y hora se publicará en el campus virtual.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Utilizando exclusivamente las operaciones constructoras generadoras de los árboles binarios, definir la sintaxis y la semántica de la operación *sumar* que crea un árbol binario resultado de sumar las etiquetas de los nodos simétricos de 2 árboles binarios dados cuyas etiquetas son números naturales.

Nota: Si un nodo de un árbol no tiene nodo simétrico en el otro, en el árbol binario resultado no aparecerá dicho nodo.



2. **PARTE A.** El siguiente fragmento de código corresponde a la definición de la clase *TDir*, que representa a una DIRECCIÓN INTERNET y está básicamente compuesta de un NOMBRE y una DIRECCIÓN IP, siendo ambos datos una cadena de caracteres. El código pertenece a la definición de prototipos (*TDir.h*). Se pide:

- Indicar en qué líneas ves INCORRECCIONES desde el punto de vista de una CORRECTA PROGRAMACIÓN C++ . Pueden ser ERRORES LÉXICO-SINTÁCTICOS (producen ERROR DE COMPILACIÓN), o bien INCORRECCIONES desde el punto de vista de una correcta programación (no necesariamente producirían ERROR DE COMPILACIÓN).
- Indicar, para cada error, la palabra “COMPILA” (si NO producen ERROR DE COMPILACIÓN) o “NO COMPILA” (si producen ERROR DE COMPILACIÓN)

Ejemplo de contestación: “LÍNEA x: el error es <bla bla bla> . COMPILA”

```
1  #ifndef _TDIR_
2  #define _TDIR_
3
4  #include <iostream>
5  #include <cstring>
6  using namespace std;
7
8  class TDir
9  {
10 public:
11     TDir();
12     TDir(char *Nombre, char *IP);
13     TDir(const TDir );
14     ~TDir();
15
16 private:
17     char* Nombre;
18     char* IP;
19     friend ostream & operator<<(ostream &, Tdir &);
20
21 }.
22
23 # end
```

PARTE B. El siguiente fragmento de código corresponde a la clase *TDir*, la misma que la de la parte A. El código pertenece a la programación de los métodos (*TDir.cc*). Se pide:

- Indicar en qué líneas ves INCORRECCIONES desde el punto de vista de una CORRECTA PROGRAMACIÓN C++ . Pueden ser ERRORES LÉXICO-SINTÁCTICOS (producen ERROR DE COMPILACIÓN), o bien INCORRECCIONES desde el punto de vista de una correcta programación de la clase (no necesariamente producirían ERROR DE COMPILACIÓN). **NOTA:** si el error se debe a una OMISIÓN (es decir, FALTA código por poner), se debe señalar la LÍNEA VACÍA donde debería aparecer este código. Para ello, se han dejado líneas vacías en el código propuesto.
- Indicar, para cada error, la palabra “COMPILA” (si NO producen ERROR DE COMPILACIÓN) o “NO COMPILA” (si producen ERROR DE COMPILACIÓN)

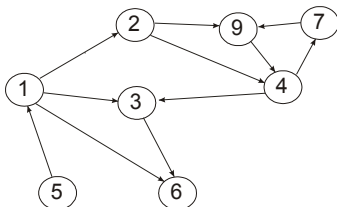
Ejemplo de contestación: “LÍNEA x: el error es <bla bla bla> . COMPILA”

```

1
2  #include <iostream>
3
4  #include <cstring>
5
6  TDir::TDir() {
7
8      Nombre=NULL;
9
10     IP=NULL;
11
12 };
13
14 TDir::TDir(const TDir &a){
15
16     Nombre=new char[strlen(a.Nombre)+1];
17
18     strcpy(Nombre,a.Nombre);
19
20     if(a->IP != NULL){
21
22     {
23
24         IP=new char[strlen(a.IP)+1];
25
26         strcpy(IP,a.IP);
27
28     }
29
30 }
31
32 bool TDir::~~TDir(){
33
34     if(Nombre != NULL){
35
36         Nombre=NULL;
37
38     }
39
40     if(IP != NULL){
41
42         delete IP;
43
44         IP=NULL;
45
46     }
47
48 }

```

3. Sea el siguiente grafo dirigido:



- Realiza el recorrido DFS(1), con la adyacencia de salida ordenada de menor a mayor, y obtén el árbol extendido en profundidad.
- Etiqueta los arcos.
- ¿Este grafo tiene ciclos? ¿Por qué?
- ¿Es un grafo fuertemente conexo? Justifica tu respuesta.

4. Calcular las cotas superiores de complejidad temporal en su caso peor, para un montículo simple máximo y un montículo doble, para las siguientes operaciones:

Insertar(elemento), Borrar(elemento), int NúmeroDeElementos(), ListarElementosPorNiveles(), int DevolverMáximo(), elemento DevolverMínimo()

Para ello suponed que la implementación interna de los montículos es como sigue:

```

class Montículo {
    int * vector;
    int numeroElementos; ... };

```

NOTA: Hay que explicar cada complejidad con un máximo de 3 líneas. Si la explicación no es correcta, no se valorará positivamente.

Examen TAD/PED diciembre 2004. Soluciones

1)

sumar: arbin, arbin → arbin

var i,l,d,r:arbin; x,y:natural;

sumar(crear_arbin(),crear_arbin()) = crear_arbin()

sumar(crear_arbin(),d) = crear_arbin()

sumar(i,crear_arbin()) = crear_arbin()

sumar(enraizar(i,x,d),enraizar(l,y,r)) = enraizar(sumar(i,l),x+y ,sumar(d,r))

2)

```
1  #ifndef _TDIR_
2  #define _TDIR_
3
4  #include <iostream>
5  #include <cstring>
6  using namespace str;
7
8  class TDir
9  {
10 public:
11     TDir();
12     TDir(char *Nombre, char *IP);
13     TDir(const TDir );           // FALTA "&"  COMPILA
14     ~TDir();
15
16 private:
17     char* Nombre;
18     char* IP;
19 friend ostream & operator<<(ostream &, Tdir &); // mayúscula : "TDir"  NO COMPILA
20
21 } // cambiar "." por ";"  NO COMPILA
22
23 #end // cambiar "end" por "endif"  NO COMPILA
```

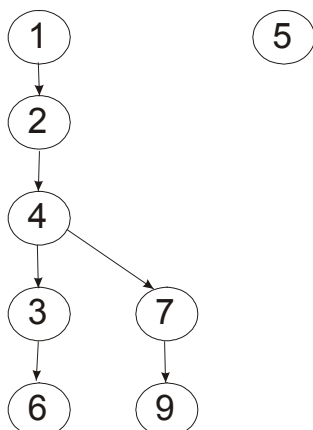
```

1 //FALTA #include "TDir.h" NO COMPILA
2 #include <iostream>
3
4 #include <cstring>
5
6 TDir::TDir() {
7
8     Nombre=NULL;
9
10    IP=NULL;
11
12 };
13
14 TDir::TDir(const TDir &a){
15     // FALTA if(a.Nombre!=NULL){ COMPILA
16     Nombre=new char[strlen(a.Nombre)+1];
17
18     strcpy(Nombre,a.Nombre);
19
20     if(a->IP != NULL){ // cambiar "->" por "." NO COMPILA
21
22     {
23
24         IP=new char[strlen(a.IP+1)]; //es "new char[strlen(a.IP)+1]" NO COMPILA
25
26         strcpy(IP,a.IP);
27
28     }
29
30 }
31
32 bool TDir::~~TDir(){ // quitar "bool" NO COMPILA
33
34     if(Nombre != NULL){
35         // FALTA "delete Nombre" COMPILA
36         Nombre=NULL;
37
38     }
39
40     if(IP != NULL){
41
42         delete IP;
43
44         IP=NULL;
45
46     }
47
48 }

```

3)

a)



b)

(1,2) = Arbol.
 (2,4) = Arbol.
 (4,3) = Arbol
 (4,7) = Arbol
 (3,6) = Arbol
 (7,9) = Arbol
 (1,3) = Avance.
 (1,6) = Avance.
 (2,9) = Avance.
 (5,1) = Cruce.
 (9,4) = Retroceso.

c)

Si tiene ciclos porque existe un arco de retroceso. (9,4)

d)

No es un grafo fuertemente conexo.

Para que no sea un grafo fuertemente conexo basta con que existan mas de una componente fuertemente conexa. La componente fuertemente conexa formada por el nodo 6 ya es una componente fuertemente conexa y como no están incluidos todos los nodos del grafo, quiere decir que hay más de una.

4)

	Montículo Simple Máximo	Montículo Doble
Insertar(elemento)	$O(\log_2 n)$ habría que realizar un recorrido ascendente: la altura de un árbol binario completo	$O(\log_2 n)$ habría que realizar recorridos ascendentes: la altura de un árbol binario completo
Borrar(elemento)	$O(\log_2 n)$ habría que realizar un recorrido descendente: la altura de un árbol binario completo	$O(\log_2 n)$ habría que realizar recorridos descendentes: la altura de un árbol binario completo
NúmeroDeElementos	$O(1)$ se accede al campo <i>numeroElementos</i>	$O(1)$ se accede al campo <i>numeroElementos</i>
ListarElementosPorNiveles	$O(n)$ habría que recorrer todo el vector hasta alcanzar los n elementos	$O(n)$ habría que recorrer todo el vector hasta alcanzar los n elementos
DevolverMáximo	$O(1)$ se accedería en la raíz del montículo que está en la primera posición del <i>vector</i>	$O(1)$ se accedería en la raíz del montículo máximo que está en la segunda posición del <i>vector</i>
DevolverMínimo	$O(n)$ siguiendo el algoritmo de recorrer todos los elementos del vector almacenando en todo momento el mínimo elemento encontrado	$O(1)$ se accedería en la raíz del montículo mínimo que está en la primera posición del <i>vector</i>

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED diciembre 2005

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - **El test vale un 40% de la nota de teoría.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Las colas también se conocen como listas FIFO.	<input type="checkbox"/>	<input type="checkbox"/>	1.	V
Un árbol binario lleno es también un árbol 2-3-4.	<input type="checkbox"/>	<input type="checkbox"/>	2.	F
Sea un árbol 2-3-4 inicialmente vacío. Tras utilizar las operaciones de inserción y borrado de un árbol 2-3-4 siempre se consigue un árbol binario lleno.	<input type="checkbox"/>	<input type="checkbox"/>	3.	F
Un árbol binario completo es un árbol 2-3-4.	<input type="checkbox"/>	<input type="checkbox"/>	4.	F
Un árbol binario completo de altura h y con $2^h - 1$ nodos es un árbol 2-3-4.	<input type="checkbox"/>	<input type="checkbox"/>	5.	F
Un árbol Rojo – Negro con todos sus enlaces negros tiene los mismos nodos que el árbol 2-3-4 equivalente.	<input type="checkbox"/>	<input type="checkbox"/>	6.	V
En un árbol B todos los nodos han de tener al menos $m/2$ hijos o $(m-1)/2$ claves.	<input type="checkbox"/>	<input type="checkbox"/>	7.	F
En una tabla Hash con dispersión cerrada, las casillas vacías hay que diferenciarlas de las suprimidas para realizar una inserción.	<input type="checkbox"/>	<input type="checkbox"/>	8.	F
En un Hash cerrado, el factor de carga ($\alpha = n/B$) puede ser mayor que 1 cuando n sea mayor que B.	<input type="checkbox"/>	<input type="checkbox"/>	9.	F
En un multigrafo pueden existir infinitas aristas para un numero “n” de vértices.	<input type="checkbox"/>	<input type="checkbox"/>	10.	V
La operación concatena es un enriquecimiento de las operaciones definidas para el tipo cola.	<input type="checkbox"/>	<input type="checkbox"/>	11.	V
Es posible reconstruir un único árbol AVL a partir de un recorrido por niveles.	<input type="checkbox"/>	<input type="checkbox"/>	12.	V
El borrado en un árbol AVL nunca requiere más de una rotación en el camino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	13.	F
En la escala de complejidades, la complejidad cuadrática es menor que la logarítmica.	<input type="checkbox"/>	<input type="checkbox"/>	14.	F

Examen PED diciembre 2005

- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: *Apellidos, Nombre*.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible.
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación de notas de exámenes:** 2 de diciembre. **Fecha de revisión de exámenes:** se publicará en el campus virtual.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Utilizando las operaciones definidas en clase para la definición del tipo vector definir la sintaxis y la semántica de la operación *inversos* que indica si dos vectores de naturales dados son inversos entre sí, es decir, si un vector es el inverso del otro. Nota: se asume que el vector está creado con 100 componentes y el rango de las mismas es de 1...100.

2. Dada la siguiente declaración de la clase TArbin que representa un árbol binario de búsqueda cuyas etiquetas son enteros y donde no se permiten etiquetas repetidas, escribe el código del constructor por defecto TArbin(), del destructor ~TArbin() y del método Insertar(int), que devuelve cierto si el número se puede insertar y falso en caso contrario. Escribe también el código del constructor y destructor de la clase TNode. Importante: se tiene que escribir el código de todos los métodos auxiliares que se empleen.

```
class TArbin {
public:
    TArbin();
    ...
    ~TArbin();
    ...
    bool Insertar(int);
    ...

private:
    TNode *n;
};

class TNode {
public:
    TNode();
    ...
    ~TNode();
    ...

private:
    int etiqueta;
    TArbin hijoI, hijoD;
};
```

3. Sea un trie (RETRIEVAL) diseñado para almacenar direcciones de memoria en formato hexadecimal inicialmente vacío y con una complejidad temporal optimizada.

- Expresa la representación de los nodos del trie para este caso concreto y explica su funcionamiento.
- Inserta en el trie las siguientes direcciones de memoria: 0001, 000B, AB00, AB01, ABCD, A001, AA11, FF00, FFF0, FF3A, DC45, 000, FF.

4. Dado el DEAP representado en forma del siguiente vector, tal y como se ha visto en clase:

<i>l</i>	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	1	80	4	7	32	53	5	10	9	8	25	31	40	50	20	15	12	19	35	11	29	30	21	17	24	27	36	28	33

- Realizar el borrado del mínimo y del máximo en este orden.
- Dado un índice *i* del vector (en el ejemplo de *a*), *i* tomará valores entre 2 y 30) que está en un nivel *h* de un DEAP con *n* nodos (en el ejemplo de *a*), *n* tomará el valor de 30), explicar cómo se obtiene:
 - El nodo padre.
 - El nodo hijo izquierdo e hijo derecho.
 - El nodo simétrico suponiendo que *i* está en el montículo máximo.
 - El nodo simétrico suponiendo que *i* está en el montículo mínimo.

Examen PED diciembre 2005. Soluciones

1.

Sintaxis:

inversos: vector, vector --> bool
invAux: vector, vector --> bool

Semántica:

Var v,w: vector; i,j,x,y: natural;

inversos(v,w) = invAux(v,w) && invAux(w,v)
invAux(crear_vector(),w) = VERDADERO
si x == recu(w,100-i+1) entonces invAux(asig(v,i,x),w) = invAux(v,w)
sino invAux(asig(v,i,x),w) = FALSO

2.

```
TArbin::TArbin() {
    n = NULL;
}

TArbin::~~TArbin() {
    if(n != NULL)
    {
        delete n;
        n = NULL;
    }
}

bool
TArbin::Insertar(int item) {
    if(n == NULL)
    {
        n = new TNode;
        if(n == NULL)
            return false;
        n->etiqueta = item;
        return true;
    }
    else
    {
        if(item < n->etiqueta)
            return (n->hijoI).Insertar(item);
        else if(item > n->etiqueta)
            return (n->hijoD).Insertar(item);
        else
            return false;
    }
}

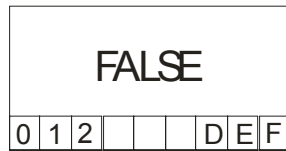
TNode::TNode() {          // Se puede dejar en blanco
    etiqueta = 0;
}

TNode::~~TNode() {        // Se puede dejar en blanco
    etiqueta = 0;
}
```

3. a) El trie tendrá dos tipos de nodos:

- Nodos internos:
Son aquellos que se encuentran en el interior de la estructura y que contiene una variable booleana y un vector de 16 punteros a nodo cada uno de los cuales servirá para identificar

cada una de los posibles valores hexadecimales (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). La representación de este nodo será la siguiente:

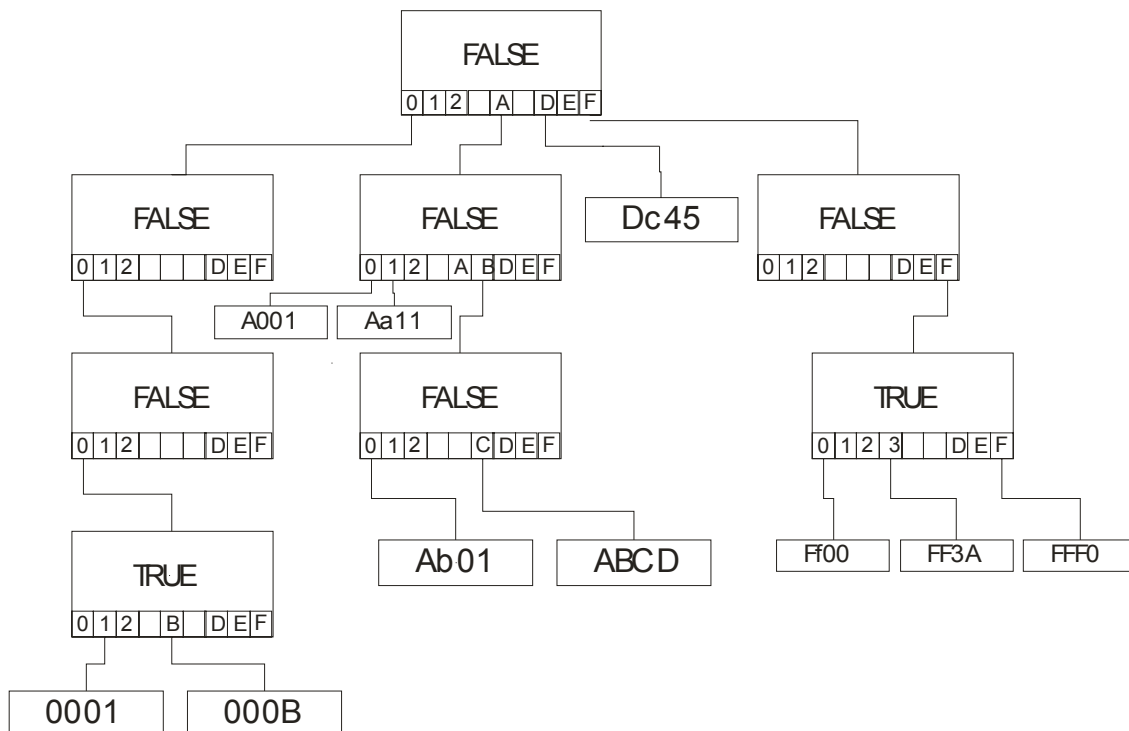


- **Nodos terminales:**
Son aquellos que se encuentran en las hojas del Trie y se utilizarán cuando no existan más de una dirección con el mismo prefijo que la que está almacenando.
La estructura de este nodo será únicamente un array para almacenar la palabra.

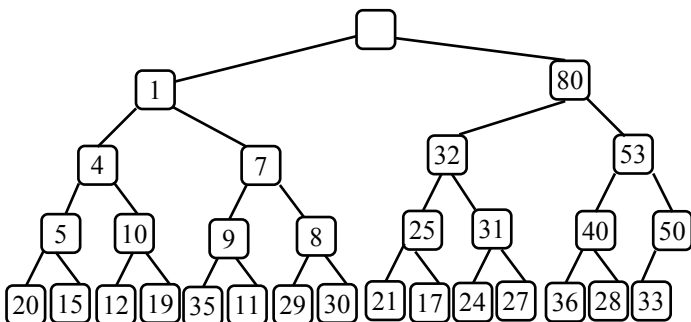
Funcionamiento:

Como se explica en los apuntes.

b)

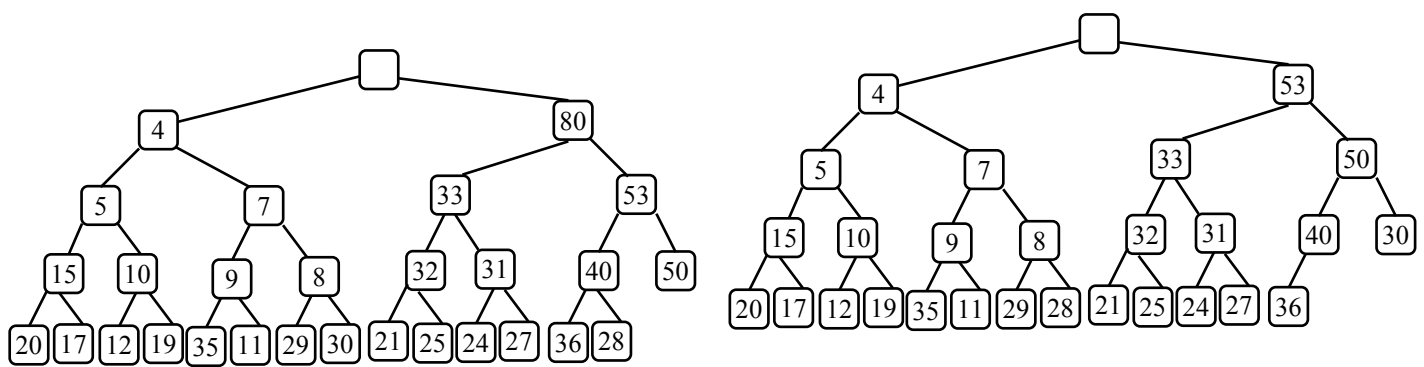


4. a) DEAP de partida, reconstruido a partir del vector:



Borrado del mínimo:

Borrado del máximo:



b)

- El nodo padre: $i \text{ DIV } 2$
- El nodo hijo izquierdo $2*i$ e hijo derecho $2*i+1$ (siempre que estos valores sean menores o iguales que n , sino no existiría el hijo).
- El nodo simétrico suponiendo que i está en el montículo máximo: sabiendo que en el nivel h del árbol hay un máximo de 2^{h-1} nodos, habría que restarle la mitad de esos nodos: $i-2^{h-2}$
- El nodo simétrico suponiendo que i está en el montículo mínimo: sabiendo que en el nivel h del árbol hay un máximo de 2^{h-1} nodos, habría que sumarle la mitad de esos nodos: $i+2^{h-2}$, y si ese valor es mayor que n entonces dividirlo por dos para retroceder al padre.

Examen PED diciembre 2007

Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
 - Tiempo para efectuar el test: **15 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes: VAR x, y: natural; suma(x, cero) = x suma(cero, x) = x suma(x, suc(y)) = suma(suc(x), y)	<input type="checkbox"/>	<input type="checkbox"/>	1	F
En C++, la siguiente declaración es INCORRECTA : const int& a = 1;	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dadas las clases TDir y TVectorDir con todos sus métodos implementados: constructor, constructor sobrecargado, sobrecarga del corchete, sobrecarga del operador salida (se muestra el contenido de cada posición del vector dejando un espacio), etc. Nota: El constructor por defecto crea un vector vacío. El constructor a partir de una dimensión, pone todos los elementos a 0.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
<pre> class TDir class TVectorDir main() { public: private: int e1; int e2; }; class TVectorDir { public: private: TDir *vector; int longitud; }; main() { TDir a(1,1); TVectorDir v; cout<<"v_antes="<<v<<endl; v[1]=a; cout<<"v_despues="<<v<<endl; }</pre> El resultado obtenido tras la ejecución del <i>main()</i> sería: v_antes= 0 0 v_despues= 1 1				
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
La complejidad temporal de un algoritmo depende de la complejidad espacial del mismo.	<input type="checkbox"/>	<input type="checkbox"/>	5	F
La semántica de la operación <i>desencolar</i> vista en clase es la siguiente: VAR c: cola, x: item; si esvacía(c) entonces desencolar(encolar(c, x)) = crear_cola () si no desencolar(encolar(c, x)) = encolar(desencolar (c), x)	<input type="checkbox"/>	<input type="checkbox"/>	6	V
Un árbol con un único nodo es un árbol lleno.	<input type="checkbox"/>	<input type="checkbox"/>	7	V
Un árbol con un único nodo tiene un único camino cuya longitud es 1.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.	<input type="checkbox"/>	<input type="checkbox"/>	9	V
En la representación de conjuntos mediante listas, la complejidad espacial es proporcional al tamaño del conjunto representado.	<input type="checkbox"/>	<input type="checkbox"/>	10	V
En un grafo dirigido pueden existir infinitas aristas para un número "n" de vértices.	<input type="checkbox"/>	<input type="checkbox"/>	11	F

Examen PED diciembre 2007

- Normas:**
- Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** se publicará un anuncio en el campus virtual.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Define la sintaxis y semántica de la operación **posición** que actúa sobre un vector y devuelve la posición menor sobre la que se ha asignado el valor que recibe como parámetro. Si no se ha asignado el valor en el vector se debe devolver 0. Ejemplo:

`posición(asig(crear(), 3, b), a) = 0`

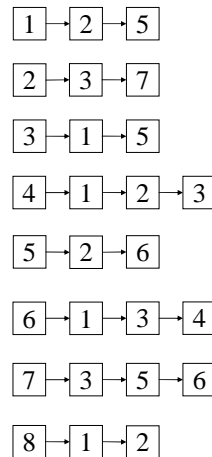
`posición(asig(asig(crear(), 3, b), 1, b), b) = 1`

`posición(asig(asig(asig(crear(), 3, b), c, 2), 1, b), b) = 1`

`posición(asig(asig(asig(crear(), 1, b), c, 2), 3, b), b) = 1`

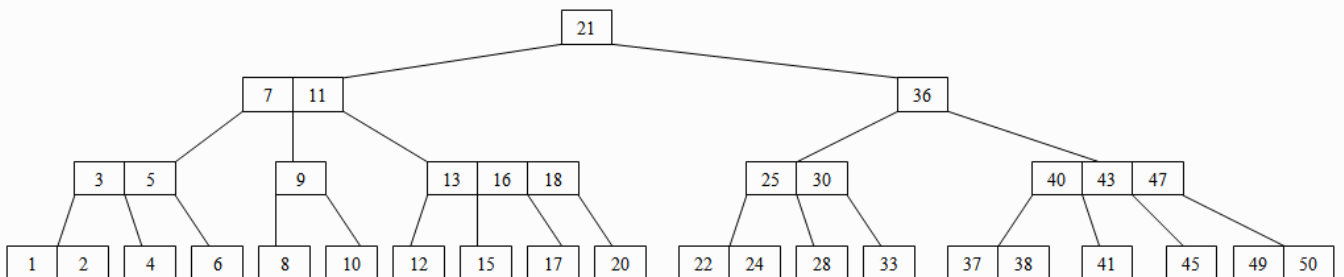
2. Dado el siguiente grafo no dirigido representado por la lista de adyacencia que se muestra a continuación:

- Realiza el recorrido DFS(1), recorriendo la adyacencia de menor a mayor y obtén el bosque extendido en profundidad.
- Calcula las componentes fuertemente conexas del grafo inicial.

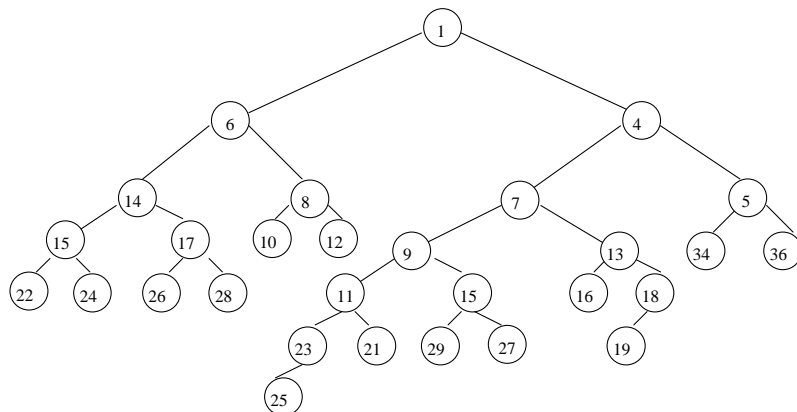


3. Sobre el siguiente árbol 2-3-4, realizar los siguientes borrados consecutivos: 25, 11, 21. Criterios:

- Sustituir por el mayor de la izquierda
- Si existen dos nodos adyacentes escoger 'r' hermano de la derecha



4. Dados el siguiente árbol leftist mínimo, realizar dos borrados sucesivos, y sobre el resultado final, la inserción de la clave 20.



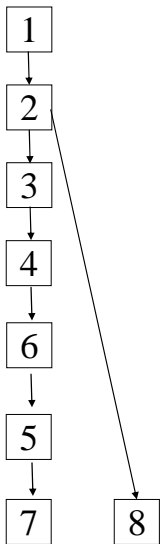
Examen PED diciembre 2007. Soluciones

1.

```
posicion(vector, item) → entero
posicionAux(vector, item, entero) → entero
Var v:vector; i: entero; x: item;
posicion(crear(), x) = 0
si (x <> y) entonces
    posicion(asig(v, i, x), y) = posicion(v, y)
si no posicion(asig(v, i, x), y) = posicionAux(v, y, i)
posicionAux(crear(), x, i) = i
si (x <> y) entonces
    posicionAux(asig(v, i, x), y, j) = posicionAux(v, y, j)
si no si (i < j) entonces
    posicionAux(asig(v, i, x), y, j) = posicionAux(v, y, i)
    si no posicionAux(asig(v, i, x), y, j) = posicionAux(v, y, j)
```

2.

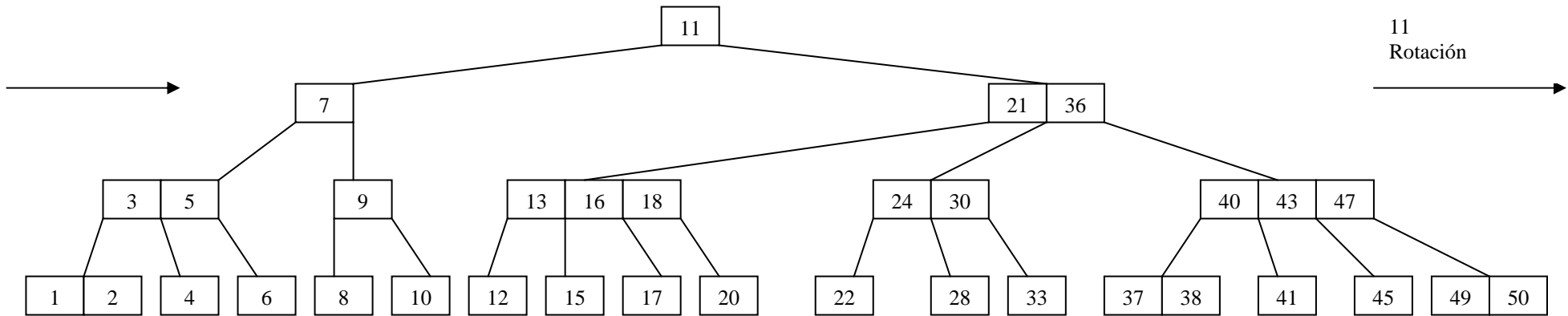
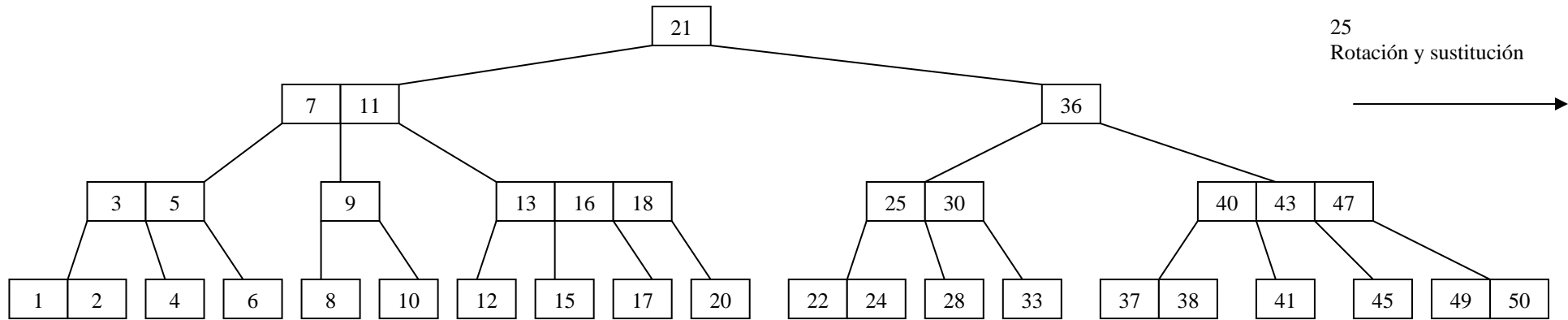
a) DFS(1): 1,2,3,4,6,5,7,8

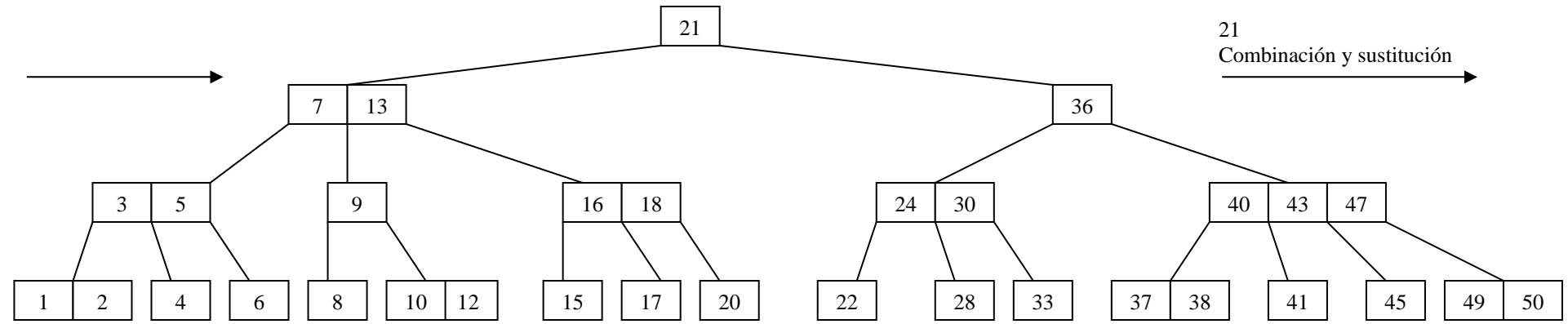
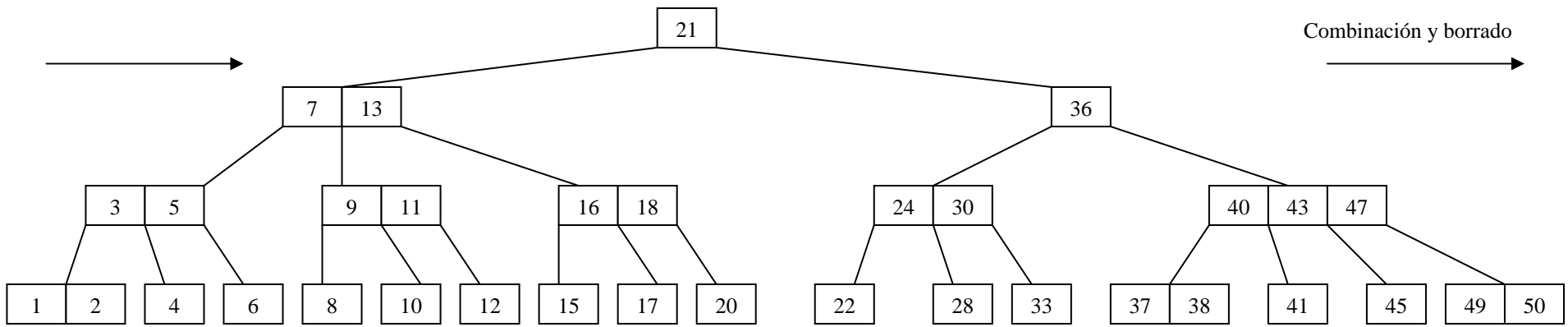
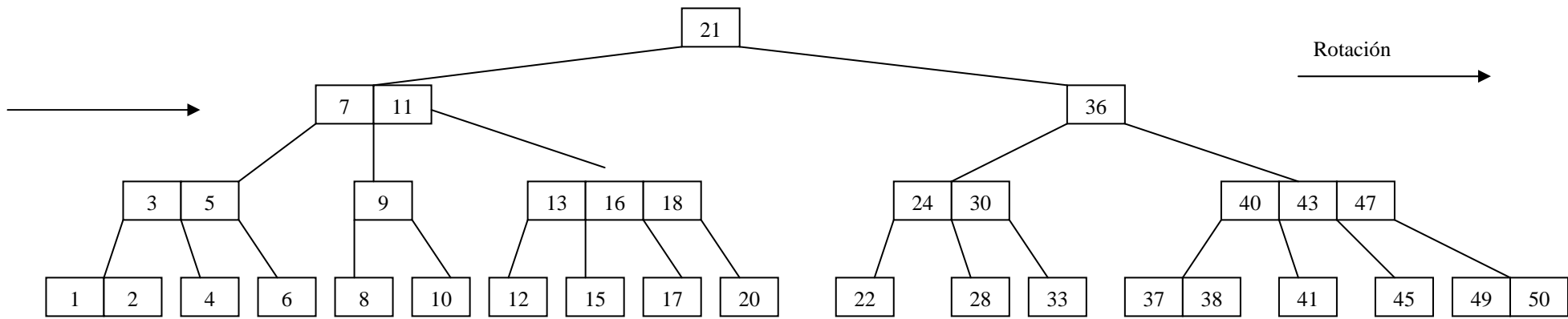


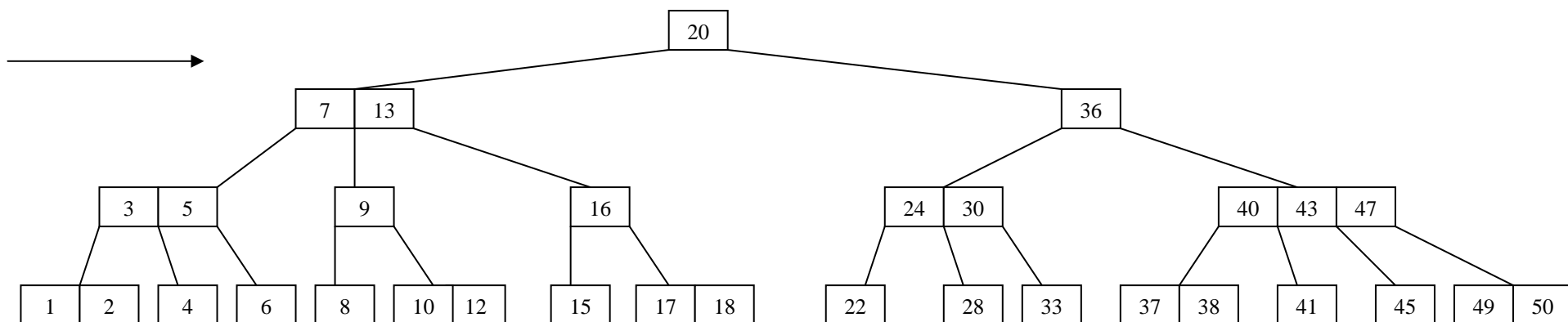
b) Componentes fuertemente conexas: {1,2,3,4,5,6,7,8}

3.

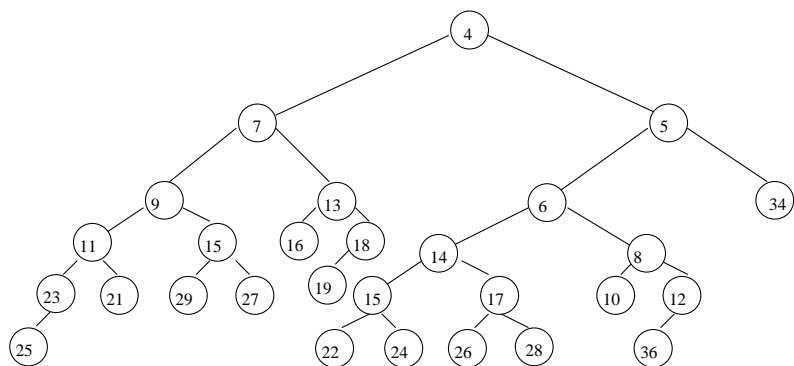
Solución:



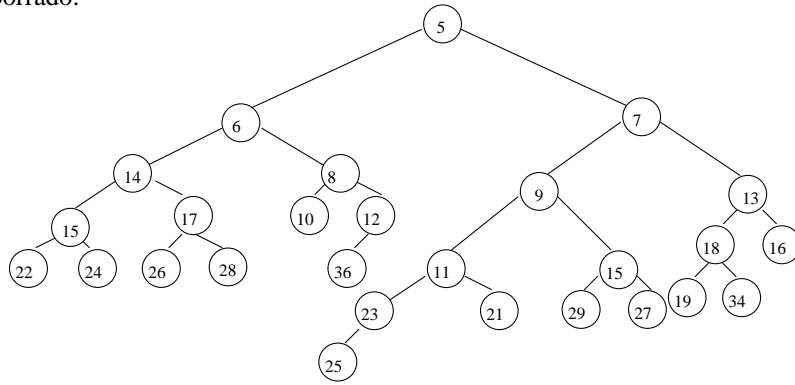




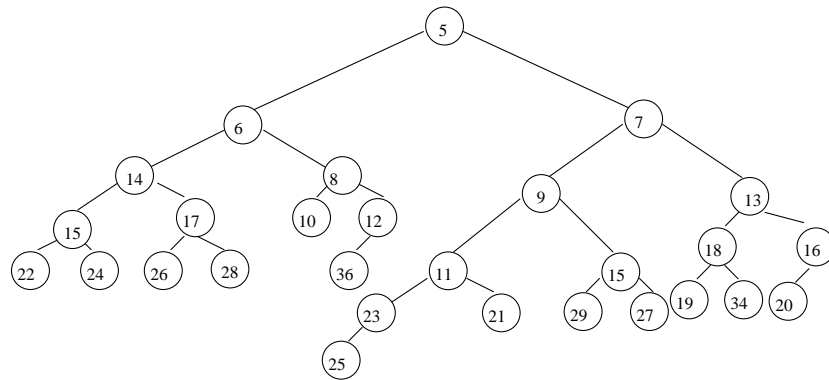
4. Primer borrado:



Segundo borrado:



Inserción del 20:



DNI:

Normas:

- La entrega del test no corre convocatoria.
- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca: $TPosicion\ TLista::Primera() \quad \text{class } TLista \{$ $\{ \quad TPosicion\ p; \quad \quad \quad \text{public: } \dots$ $\quad \quad \quad \text{p.pos} = lis; \quad \quad \quad \text{private:}$ $\quad \quad \quad \text{return } p; \} \quad \quad \quad \text{TNode } *lis; \}$	<input type="checkbox"/>	<input type="checkbox"/>	1	F
En la línea resaltada, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.				
En la escala de complejidades se cumple que $O(n \log n) \subset O(n^2)$.	<input type="checkbox"/>	<input type="checkbox"/>	2	V
La operación BorrarResultem tiene la siguiente sintaxis y semántica: BorrarResultem: LISTA, ITEM -> LISTA BorrarResultem(Crear, i) = Crear BorrarResultem(IC(L1,j), i) = si (i == j) entonces L1 sino IC (BorrarResultem (L1, i), j)	<input type="checkbox"/>	<input type="checkbox"/>	3	V
Esta operación borra la primera ocurrencia del ítem que se encuentra en la lista				
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input type="checkbox"/>	4	F
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	5	F
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, sólo se va a efectuar una rotación como mucho	<input type="checkbox"/>	<input type="checkbox"/>	6	V
Dado un árbol 2-3 con n ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 n)$	<input type="checkbox"/>	<input type="checkbox"/>	7	V
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Un árbol rojo-negro es un árbol binario balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	9	F
La raíz del árbol B m-camino de búsqueda siempre tiene al menos $m/2$ claves o etiquetas	<input type="checkbox"/>	<input type="checkbox"/>	10	F
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem): $Eliminar(Crear, x) \Leftrightarrow Crear$ $Eliminar(Insertar(C, x), y) \Leftrightarrow$ $si\ (x == y)\ \text{entonces } C\ \text{sino } Insertar(Eliminar(C, y), x)$	<input type="checkbox"/>	<input type="checkbox"/>	11	F
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $h_i(14) = (28 + 7 \cdot i) \text{ MOD } 2000$, se recorrerán todas las posiciones de la tabla buscando una posición libre.	<input type="checkbox"/>	<input type="checkbox"/>	12	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.	<input type="checkbox"/>	<input type="checkbox"/>	13	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.	<input type="checkbox"/>	<input type="checkbox"/>	14	V
La altura máxima de un árbol de búsqueda digital es “ $n+1$ ”, siendo n el número de bits de la clave.	<input type="checkbox"/>	<input type="checkbox"/>	15	V

Examen PED diciembre 2010

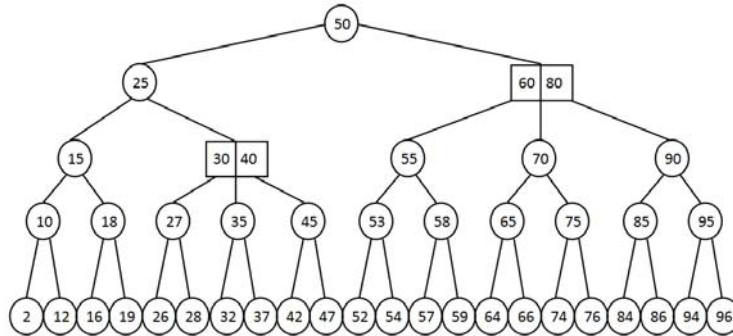
- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

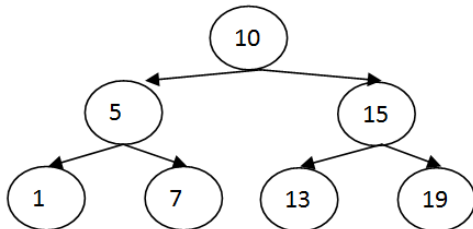
1. Realizar la especificación algebraica de una función que determine si un árbol binario cumple las condiciones de balanceo de un AVL. Para ello, se supone que están definidas las siguientes operaciones:

- Resta de naturales: $resta(a,b)$, a debe ser mayor o igual que b .
- Comparación de naturales: $\leq, \geq, =, >, <$
- Operaciones booleanas: AND, OR.
- Y todas las operaciones del tipo de datos árbol binario.

2. Borrar en el siguiente árbol 2-3 los elementos: 50, 2 y 80. Criterios: (1) si el nodo tiene dos hijos hay que sustituir por el mayor de la izquierda, (2) si el 2-nodo tiene dos hermanos, consultar el hermano de la izquierda. Nota: en el borrado de cada elemento, indicar las operaciones (transformaciones) realizadas:



3. Sea el siguiente árbol AVL:



- Insertar los ítems 1,2,4,8,11,12.
- ¿El árbol resultante del apartado a) es un leftist? Explica porqué.
- Si consideras que no es un leftist, realiza los mínimos cambios imprescindibles para convertir el árbol resultante del apartado a) en un leftist. Explica las acciones que tomes.

4. Dar **razonadamente** las complejidades temporales en el peor y mejor caso de las siguientes operaciones en función del parámetro n . Se exigirá que la justificación de la complejidad sea correcta:

- Inserción en un árbol B con $m=5$ y n elementos.
- Búsqueda de un elemento en una lista ordenada de n elementos.
- Borrado de un elemento en un montículo con n elementos.

Examen PED diciembre 2010. Soluciones

1.

VAR i,d: arbin; x: item

ES_AVL(arbin) \rightarrow bool

ES_AVL(crear()) = V

ES_AVL(enraizar(crear_arbin(), x, crear_arbin()))=V

ES_AVL(enraizar(i, x, d)) =

si altura(i) \geq altura(d)

entones

 si resta(altura(i), altura(d)) \geq suc(suc(cero))

 entones

 F

 sino

 ES_AVL(i) y ES_AVL(d)

 fsi

sino

 si resta(altura(d), altura(i)) \geq suc(suc(cero))

 entones

 F

 sino

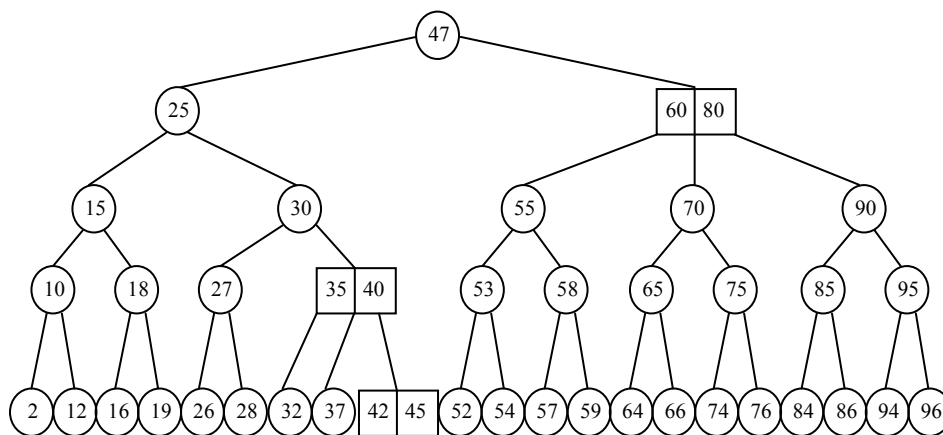
 ES_AVL(i) y ES_AVL(d)

 fsi

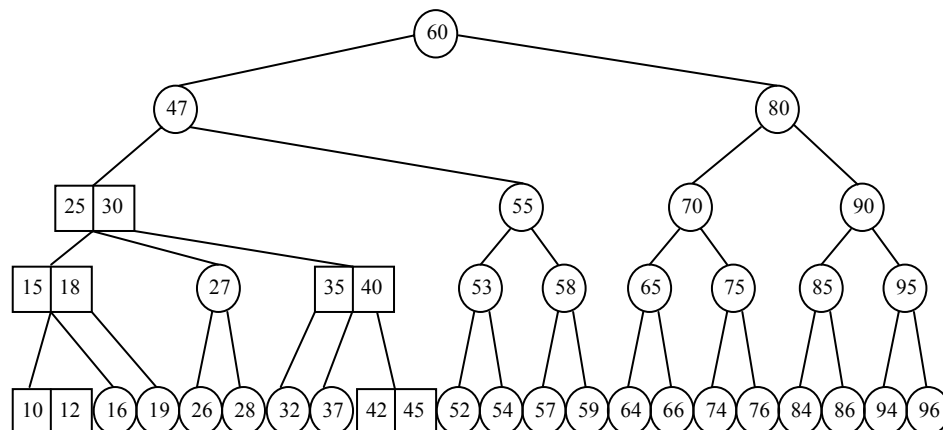
fsi

2.

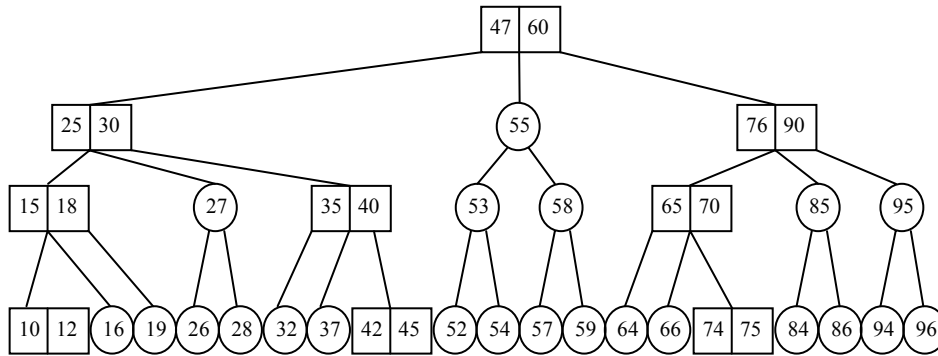
Borrado del 50 \rightarrow 2 combinaciones



Borrado del 2 \rightarrow 3 combinaciones y 1 rotación

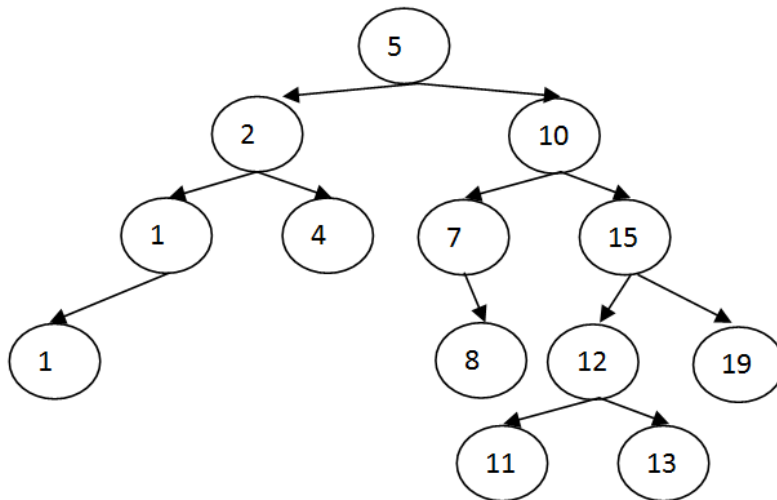


Borrado del 80 → 4 combinaciones



3.

a) Hay que tener en cuenta que el 1 ya existe. Según la propiedad de camino de búsqueda, el ítem del hijo izquierda tiene que ser menor o igual que el ítem de la raíz y la raíz menor o igual que el ítem del hijo derecha. Es por ello que en el caso de repetidos, la búsqueda de un repetido sigue por la izquierda y no por la derecha.



b) No es leftist, porque el nodo 7 no cumple la condición de leftist ni el nodo 10.

c) Para convertirlo en leftist con las menos acciones posibles, hay que intercambiar los hijos del nodo 7 y los hijos del nodo 10.

4.

- $O(\log_3(n))$ con n el número de elementos, ya que se ha de recorrer el árbol como máximo desde la raíz hasta las hojas y nuevamente hasta la raíz, con lo que la complejidad queda en función de la altura que será máxima en su caso peor cuando todos los nodos del árbol sean 3-nodo. $\Omega(\log_3(n))$, con la misma explicación anterior, salvo que sólo se realizará el recorrido descendente y la altura será mínima cuando todos los nodos estén llenos (5-nodo).
- $O(n)$, cuando el elemento a buscar esté en la última posición de la lista; y $\Omega(1)$ cuando lo encuentre en el primer intento.
- $O(\log_2(n))$ y $\Omega(\log_2(n))$, ya que siempre habrá que recorrer un camino descendente desde la raíz hasta las hojas del montículo.

Examen PED diciembre 2007

Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
 - Tiempo para efectuar el test: **15 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes: VAR x, y: natural; suma(x, cero) = x suma(cero, x) = x suma(x, suc(y)) = suma(suc(x), y)	<input type="checkbox"/>	<input type="checkbox"/>	1	F
En C++, la siguiente declaración es INCORRECTA : const int& a = 1;	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dadas las clases TDir y TVectorDir con todos sus métodos implementados: constructor, constructor sobrecargado, sobrecarga del corchete, sobrecarga del operador salida (se muestra el contenido de cada posición del vector dejando un espacio), etc. Nota: El constructor por defecto crea un vector vacío. El constructor a partir de una dimensión, pone todos los elementos a 0.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
<pre> class TDir class TVectorDir main() { public: private: int e1; int e2; }; class TVectorDir { public: private: TDir *vector; int longitud; }; main() { TDir a(1,1); TVectorDir v; cout<<"v_antes="<<v<<endl; v[1]=a; cout<<"v_despues="<<v<<endl; } </pre> El resultado obtenido tras la ejecución del <i>main()</i> sería: v_antes= 0 0 v_despues= 1 1	<input type="checkbox"/>	<input type="checkbox"/>	4	V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.	<input type="checkbox"/>	<input type="checkbox"/>	5	F
La complejidad temporal de un algoritmo depende de la complejidad espacial del mismo.	<input type="checkbox"/>	<input type="checkbox"/>	6	V
La semántica de la operación <i>desencolar</i> vista en clase es la siguiente: VAR c: cola, x: item; si esvacia(c) entonces desencolar(encolar(c, x)) = crear_cola () si no desencolar(encolar(c, x)) = encolar(desencolar(c), x)	<input type="checkbox"/>	<input type="checkbox"/>	7	V
Un árbol con un único nodo es un árbol lleno.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Un árbol con un único nodo tiene un único camino cuya longitud es 1.	<input type="checkbox"/>	<input type="checkbox"/>	9	V
Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.	<input type="checkbox"/>	<input type="checkbox"/>	10	V
En la representación de conjuntos mediante listas, la complejidad espacial es proporcional al tamaño del conjunto representado.	<input type="checkbox"/>	<input type="checkbox"/>	11	F
En un grafo dirigido pueden existir infinitas aristas para un número "n" de vértices.	<input type="checkbox"/>	<input type="checkbox"/>		

Examen PED diciembre 2007

- Normas:**
- Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** se publicará un anuncio en el campus virtual.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Define la sintaxis y semántica de la operación **posición** que actúa sobre un vector y devuelve la posición menor sobre la que se ha asignado el valor que recibe como parámetro. Si no se ha asignado el valor en el vector se debe devolver 0. Ejemplo:

`posición(asig(crear(), 3, b), a) = 0`

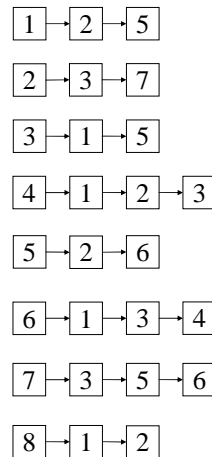
`posición(asig(asig(crear(), 3, b), 1, b), b) = 1`

`posición(asig(asig(asig(crear(), 3, b), c, 2), 1, b), b) = 1`

`posición(asig(asig(asig(crear(), 1, b), c, 2), 3, b), b) = 1`

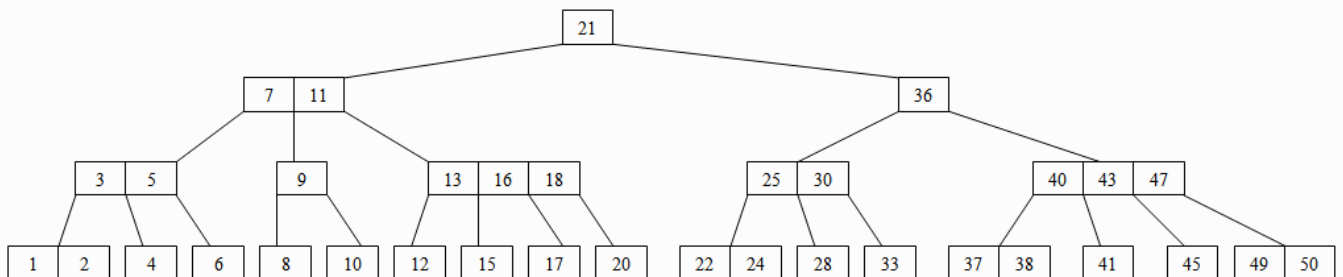
2. Dado el siguiente grafo no dirigido representado por la lista de adyacencia que se muestra a continuación:

- Realiza el recorrido DFS(1), recorriendo la adyacencia de menor a mayor y obtén el bosque extendido en profundidad.
- Calcula las componentes fuertemente conexas del grafo inicial.

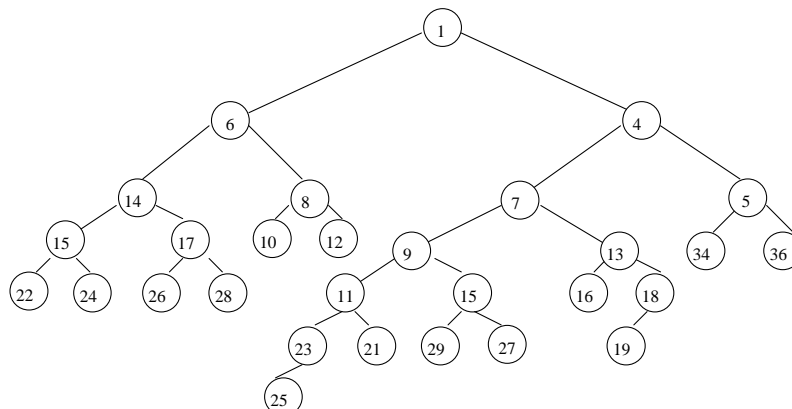


3. Sobre el siguiente árbol 2-3-4, realizar los siguientes borrados consecutivos: 25, 11, 21. Criterios:

- Sustituir por el mayor de la izquierda
- Si existen dos nodos adyacentes escoger 'r' hermano de la derecha



4. Dados el siguiente árbol leftist mínimo, realizar dos borrados sucesivos, y sobre el resultado final, la inserción de la clave 20.



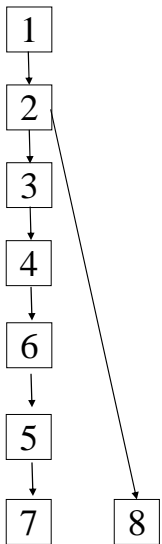
Examen PED diciembre 2007. Soluciones

1.

```
posicion(vector, item) → entero
posicionAux(vector, item, entero) → entero
Var v:vector; i: entero; x: item;
posicion(crear(), x) = 0
si (x <> y) entonces
    posicion(asig(v, i, x), y) = posicion(v, y)
si no posicion(asig(v, i, x), y) = posicionAux(v, y, i)
posicionAux(crear(), x, i) = i
si (x <> y) entonces
    posicionAux(asig(v, i, x), y, j) = posicionAux(v, y, j)
si no si (i < j) entonces
    posicionAux(asig(v, i, x), y, j) = posicionAux(v, y, i)
    si no posicionAux(asig(v, i, x), y, j) = posicionAux(v, y, j)
```

2.

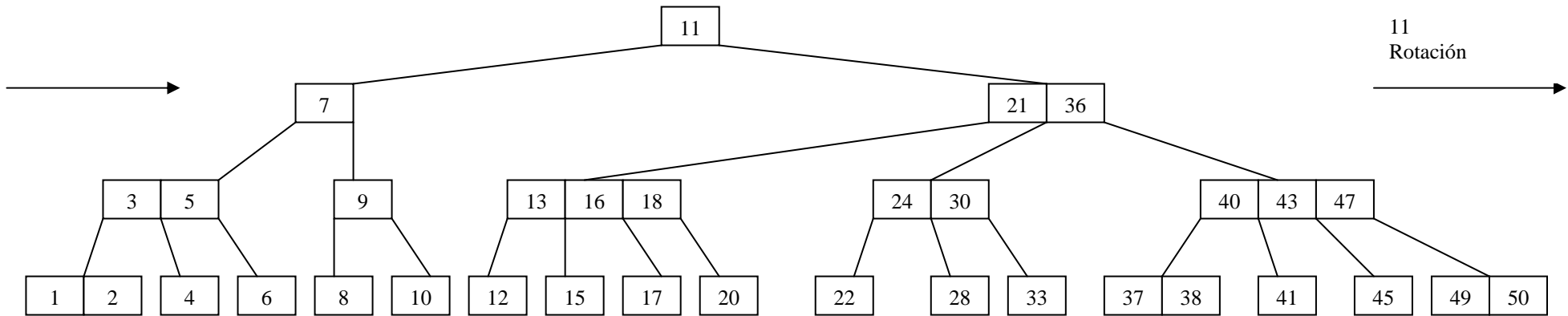
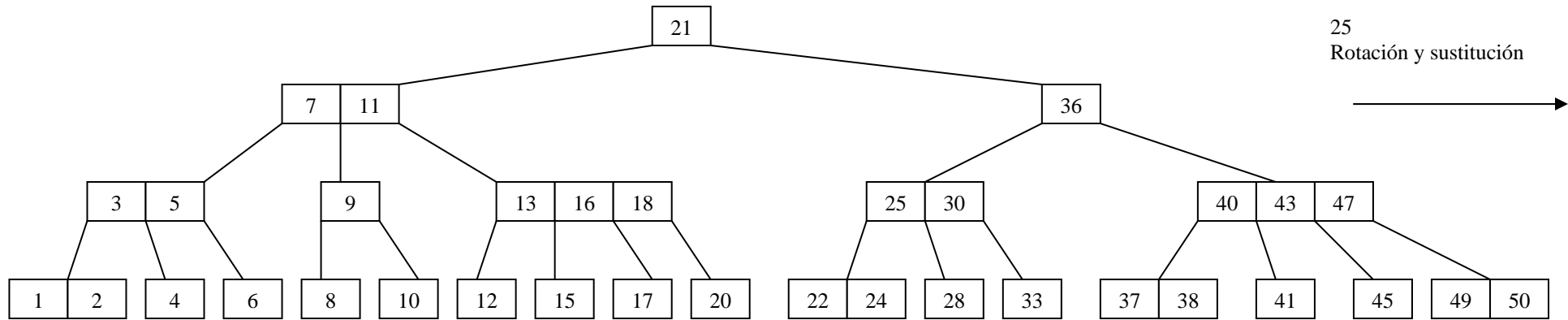
a) DFS(1): 1,2,3,4,6,5,7,8

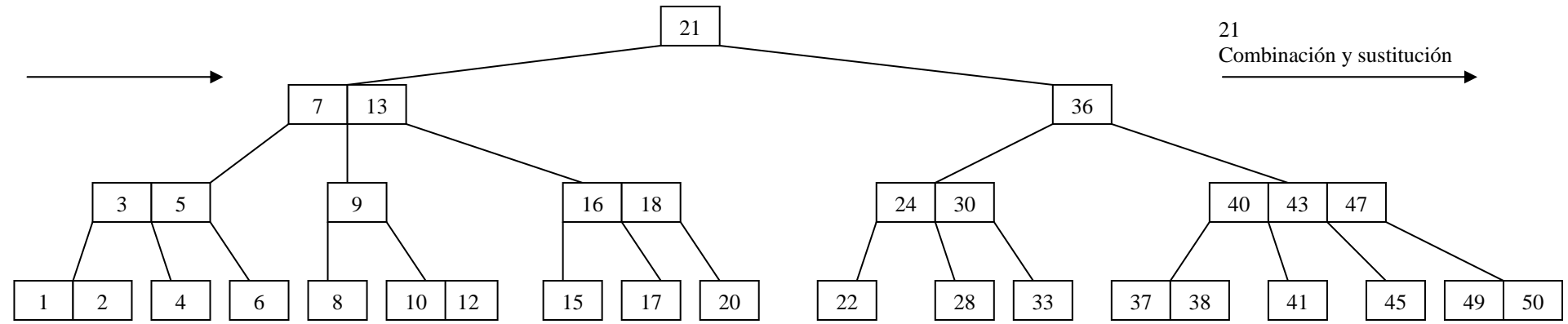
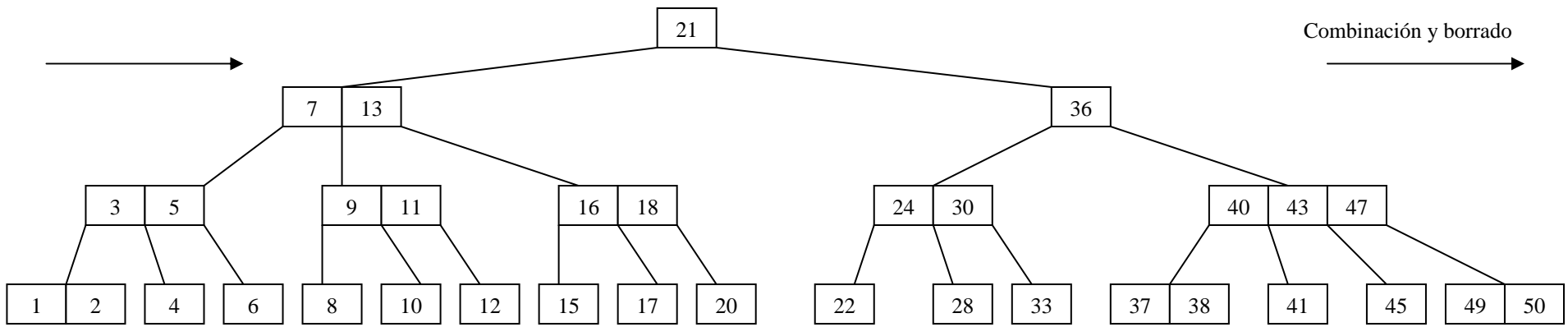
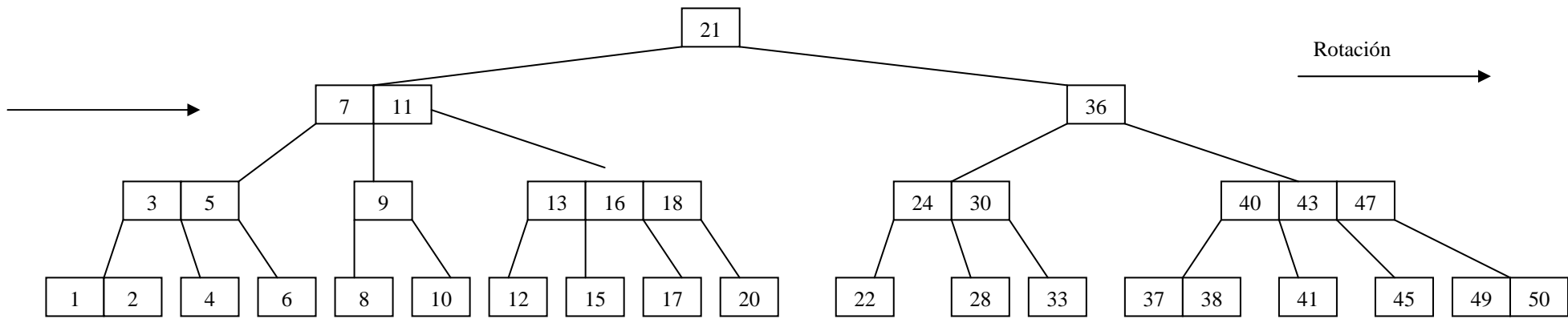


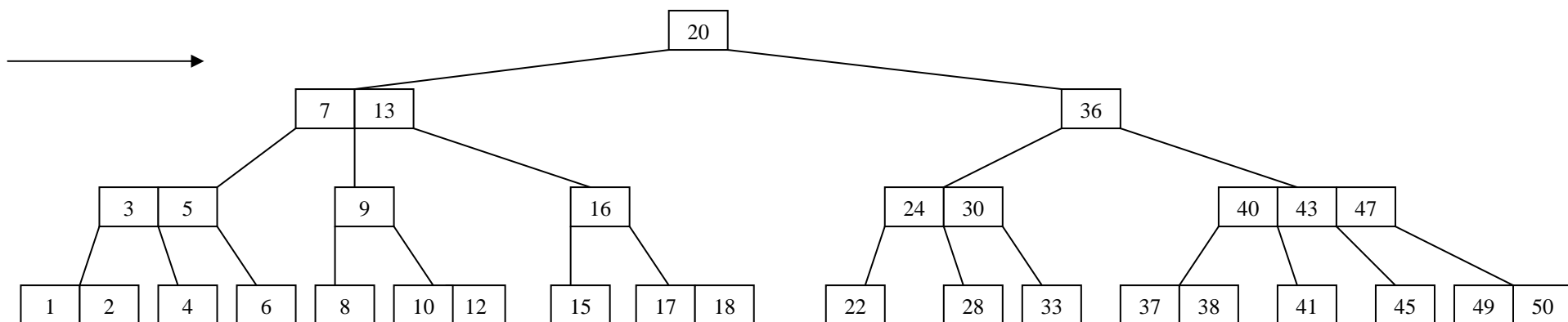
b) Componentes fuertemente conexas: {1,2,3,4,5,6,7,8}

3.

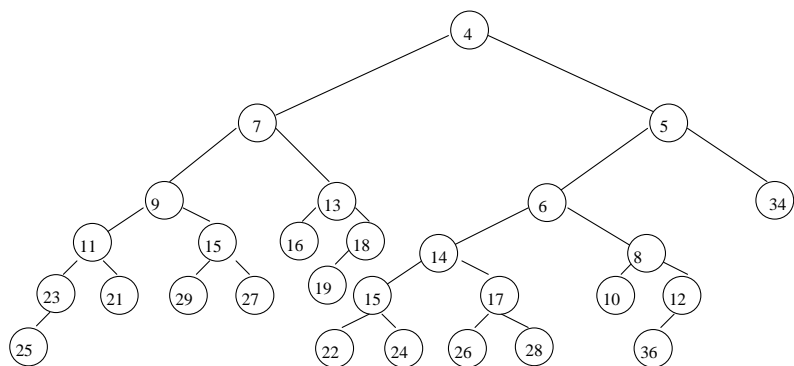
Solución:



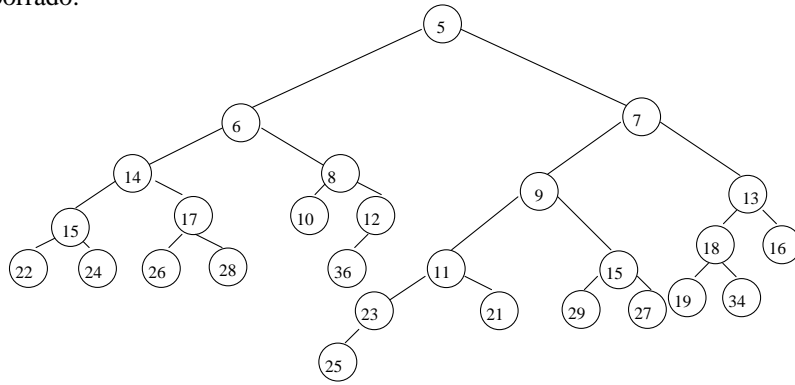




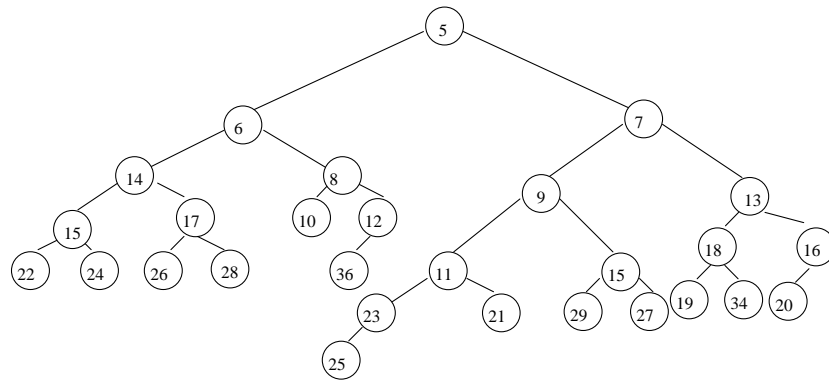
4. Primer borrado:



Segundo borrado:



Inserción del 20:



Apellidos:

Nombre:

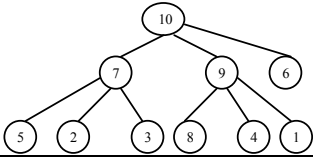
Convocatoria:

DNI:

Examen PED diciembre 2009

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **22 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Para el siguiente fragmento de código C++ de un posible método perteneciente a la conocida clase TCoordenada, la línea "delete b;" liberaría correctamente la memoria dinámica de b. <pre>void Funcion(void) { TCoordenada *a = new TCoordenada; TCoordenada *b = new TCoordenada[5]; (.....) delete b; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	1	F
El resultado del cálculo de la complejidad temporal en el mejor caso de un algoritmo X, da como resultado $n + n \cdot \log(n)$. Por lo tanto, diremos que la complejidad del algoritmo X cuando $n \rightarrow \infty$ pertenece a $\Omega(n)$.	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Las pilas también se conocen como listas LIFO.	<input type="checkbox"/>	<input type="checkbox"/>	3	V
Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
A los árboles generales también se les llama árboles multicamino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	5	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.	<input type="checkbox"/>	<input type="checkbox"/>	6	V
La altura de un árbol 2-3 únicamente crece cuando se inserta un elemento y todos los nodos del árbol son 3-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	7	F
Con las operaciones de inserción y borrado es posible conseguir un árbol 2-3-4 de altura 4 con todos sus nodos de tipo 2-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Las operaciones de transformación cuando se inserta un elemento en un árbol 2-3-4, en el caso de un árbol rojo-negro, se reducen a cambios de colores o rotaciones.	<input type="checkbox"/>	<input type="checkbox"/>	9	V
El árbol 2-3 es un árbol B m-camino de búsqueda con $m=2$.	<input type="checkbox"/>	<input type="checkbox"/>	10	F
La dispersión abierta elimina el problema del clustering secundario.	<input type="checkbox"/>	<input type="checkbox"/>	11	V
Sea una tabla de dispersión cerrada con estrategia de redistribución $h_i(x) = (H(x) + C \cdot i) \text{ MOD } B$, con $B=1000$ y $C=74$. Para cualquier clave "x" se recorrerán todas las posiciones de la tabla buscando una posición libre cuando se inserta el elemento.	<input type="checkbox"/>	<input type="checkbox"/>	12	F
El siguiente árbol es un montículo máximo: 	<input type="checkbox"/>	<input type="checkbox"/>	13	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.	<input type="checkbox"/>	<input type="checkbox"/>	14	F
Un grafo no dirigido de n vértices es un árbol si está libre de ciclos y tiene $n-1$ aristas.	<input type="checkbox"/>	<input type="checkbox"/>	15	V

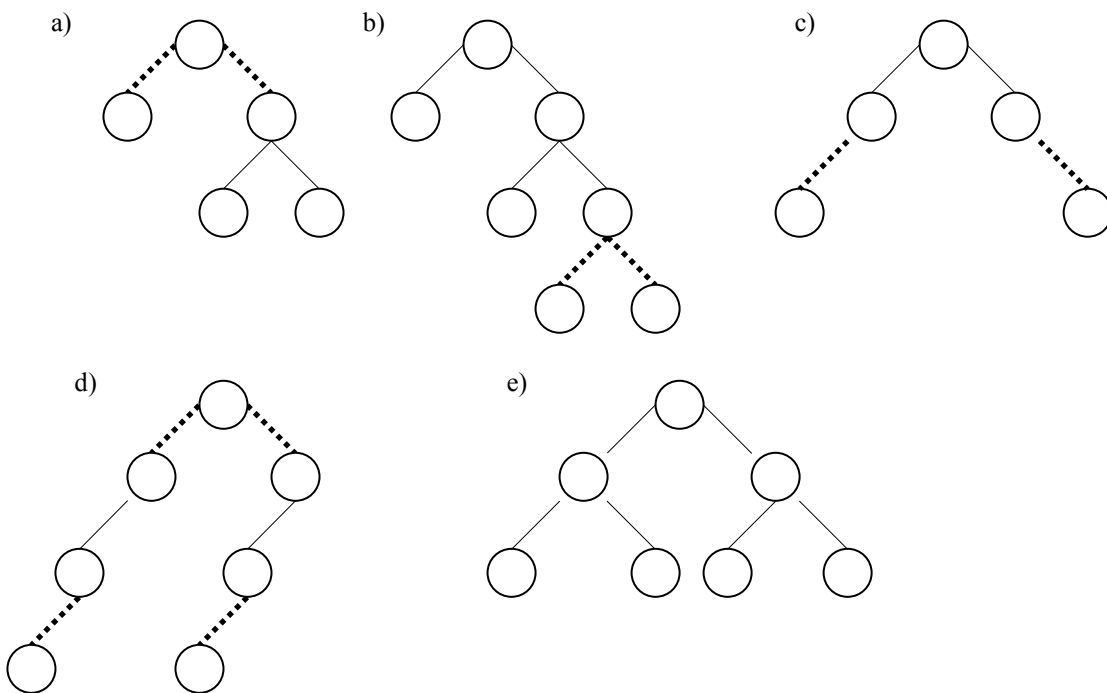
Examen PED diciembre 2009

- Normas:**
- Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** el jueves 5 de noviembre. Revisión de exámenes el martes 10 de noviembre de 9:30 a 10:30 en la sala de reuniones 'Claude Shannon' (sótano de la EPS IV). Examen de prácticas el jueves 12 de noviembre de 15:00 a 17:00 en los laboratorios L25 y L27 de la EPS I.
- Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. A partir de la especificación algebraica de la cola, escribe la sintaxis y semántica de la operación $M()$ que recibe dos colas y devuelve una cola nueva en la que se han encolado de forma alternada los elementos de las dos colas, empezando por la primera cola. Por ejemplo:

$$C1 = (a, b, c, d) \quad C2 = (1, 2, 3) \\ M(C1, C2) = (a, 1, b, 2, c, 3, d)$$

2. De los siguientes árboles determinar cuáles son R-N razonando la respuesta:



3. Insertar en una tabla de dispersión cerrada de tamaño $B=7$ los siguientes elementos: 10, 3, 17, 23, 21, 29 y 28. Mostrar el cálculo para insertar cada elemento y la tabla final con la inserción de todos los elementos:

- Con estrategia de redistribución aleatoria ($c=2$).
- Con estrategia de redistribución *segunda función hash*.

4. Escribe en C++ la forma canónica de las clases de un trie (junto con las clases de los objetos que éste contenga) que tiene las siguientes complejidades para la operación de búsqueda de una palabra: $\Omega(1)$ y $O(n * L)$, siendo n el número de letras diferentes que pueden formar parte de una palabra (por ejemplo las letras del alfabeto y los números), y L la longitud máxima de una palabra (por ejemplo de la palabra "maxilofacial", $L=12$).

NOTA:

- Hay que explicar en forma de comentarios cada campo privado de la clase.
- En la representación propuesta, explicar cuándo se producen los casos de complejidad mínima y máxima.
- El código C++ que se presente ha de ser directamente compilable, es decir, los errores de sintaxis de C++ se puntuarán negativamente.

Examen PED diciembre 2009. Soluciones

1.

**** Sintaxis:**

$M(\text{cola}, \text{cola}) \rightarrow \text{cola}$

$\text{Aux1}(\text{cola}, \text{cola}, \text{cola}) \rightarrow \text{cola}$

$\text{Aux2}(\text{cola}, \text{cola}, \text{cola}) \rightarrow \text{cola}$

**** Semántica:**

Var $c, c1, c2$: cola; x : ítem;

$M(c1, c2) = \text{Aux1}(\text{crear}(), c1, c2)$

$\text{Aux1}(c, \text{crear}(), \text{crear}()) = c$

$\text{Aux1}(c, \text{encolar}(c1, x), \text{crear}()) = \text{Aux1}(\text{encolar}(c, \text{cabeza}(c1)), \text{desencolar}(\text{encolar}(c1, x)), \text{crear}())$

$\text{Aux1}(c, \text{crear}(), \text{encolar}(c1, x)) = \text{Aux1}(\text{encolar}(c, \text{cabeza}(c1)), \text{crear}(), \text{desencolar}(\text{encolar}(c1, x)))$

$\text{Aux1}(c, \text{encolar}(c1, x), c2) = \text{Aux2}(\text{encolar}(c, \text{cabeza}(c1)), \text{desencolar}(\text{encolar}(c1, x)), c2)$

$\text{Aux2}(c, \text{crear}(), \text{crear}()) = c$

$\text{Aux2}(c, \text{encolar}(c1, x), \text{crear}()) = \text{Aux2}(\text{encolar}(c, \text{cabeza}(c1)), \text{desencolar}(\text{encolar}(c1, x)), \text{crear}())$

$\text{Aux2}(c, \text{crear}(), \text{encolar}(c1, x)) = \text{Aux2}(\text{encolar}(c, \text{cabeza}(c1)), \text{crear}(), \text{desencolar}(\text{encolar}(c1, x)))$

$\text{Aux2}(c, c1, \text{encolar}(c2, x)) = \text{Aux1}(\text{encolar}(c, \text{cabeza}(c2)), c1, \text{desencolar}(\text{encolar}(c2, x)))$

Solución (otra forma):

**** Sintaxis:**

$M(\text{cola}, \text{cola}) \rightarrow \text{cola}$

$\text{Concatena}(\text{cola}, \text{cola}) \rightarrow \text{cola}$

**** Semántica:**

Var $c1, c2$: cola;

$M(\text{crear}(), c1) = c1$

$M(c1, \text{crear}()) = c1$

$M(c1, c2) = \text{Concatena}(\text{encolar}(\text{encolar}(\text{crear}(), \text{cabeza}(c1)), \text{cabeza}(c2)), M(\text{desencolar}(c1), \text{desencolar}(c2)))$

$\text{Concatena}(\text{crear}(), c1) = c1$

$\text{Concatena}(c1, \text{crear}()) = c1$

$\text{Concatena}(c1, c2) = \text{concatena}(\text{encolar}(c1, \text{cabeza}(c2)), \text{desencolar}(c2))$

2.

a) No es R-N porque no todos los caminos desde la raíz a las hojas tienen el mismo número de nodos negros

b) Mismo razonamiento que a)

c) Sí es R-N, todos los caminos desde la raíz tienen el mismo número de nodos negros

d) Sí es R-N, mismo razonamiento que b)

e) Sí es R-N, todos los nodos del 2-3-4 equivalente son de tipo 2-nodo

3.

a) $h_i(x) = (H(x) + c \cdot i) \text{ MOD } B = (h_{i-1}(x) + c) \text{ MOD } B$

$H(10) = x \text{ MOD } B = 10 \text{ MOD } 7 = 3$

$H(3) = 3 \text{ MOD } 7 = 3$

$h_1(3) = (3+2) \text{ MOD } 7 = 5$

$H(17) = 17 \text{ MOD } 7 = 3$

$h_1(17) = (3+2) \text{ MOD } 7 = 5$

$h_2(17) = (5+2) \text{ MOD } 7 = 0$

$H(23) = 23 \text{ MOD } 7 = 2$

$H(21) = 21 \text{ MOD } 7 = 0$

$h_1(21) = (0+2) \text{ MOD } 7 = 2$

$h_2(21) = (2+2) \text{ MOD } 7 = 4$

$H(29) = 29 \text{ MOD } 7 = 1$

$H(28) = 28 \text{ MOD } 7 = 0$

$h_1(28) = (0+2) \text{ MOD } 7 = 2$

$h_2(28) = (2+2) \text{ MOD } 7 = 4$

$h_3(28) = (4+2) \text{ MOD } 7 = 6$

(0) 17 (1) 29 (2) 23 (3) 10 (4) 21 (5) 3 (6) 28

b) $k(x) = (x \text{ MOD } (B-1)) + 1 // h_i(x) = (H(x) + k(x) \cdot i) \text{ MOD } B = (h_{i-1}(x) + k(x)) \text{ MOD } B$

$H(10) = x \text{ MOD } B = 10 \text{ MOD } 7 = 3$

$$\begin{aligned}
H(3) &= 3 \bmod 7 = 3 \\
K(3) &= (3 \bmod 6) + 1 = 4 \\
h_1(3) &= (3+4) \bmod 7 = \mathbf{0} \\
H(17) &= 17 \bmod 7 = 3 \\
K(17) &= (17 \bmod 6) + 1 = 6 \\
h_1(17) &= (3+6) \bmod 7 = \mathbf{2} \\
H(23) &= 23 \bmod 7 = 2 \\
K(23) &= (23 \bmod 6) + 1 = 6 \\
h_1(23) &= (2+6) \bmod 7 = \mathbf{1} \\
H(21) &= 21 \bmod 7 = 0 \\
K(21) &= (21 \bmod 6) + 1 = 4 \\
h_1(21) &= (0+4) \bmod 7 = \mathbf{4} \\
H(29) &= 29 \bmod 7 = 1 \\
K(29) &= (29 \bmod 6) + 1 = 6 \\
h_1(29) &= (1+6) \bmod 7 = 0 \\
h_2(29) &= (0+6) \bmod 7 = \mathbf{6} \\
H(28) &= 28 \bmod 7 = 0 \\
K(28) &= (28 \bmod 6) + 1 = 5 \\
h_1(28) &= (0+5) \bmod 7 = \mathbf{5} \\
(0) \ 3 \ (1) \ 23 \ (2) \ 17 \ (3) \ 10 \ (4) \ 21 \ (5) \ 28 \ (6) \ 29
\end{aligned}$$

4.

Caso $\Omega(1)$: al buscar la palabra “zoo” si en el trie solo hay almacenada esa palabra que empiece por “z”

Caso $O(n*L)$: caso en que se busque una palabra de longitud L y en cada nodo haya que recorrer los n nodos de cada lista de strings.

```
class TNodeTrie;
```

```
class TListNode {
```

```
public:
```

```
    // Constructor por defecto
    TListNode();
```

```
    // Constructor de copia
    TListNode(const TListNode &);
```

```
    // Destructor
    ~TListNode();
```

```
    // Sobrecarga del operador asignación
    TListNode & operator=(const TListNode &);
```

```
private:
```

```
    // Letra o string a la que se corresponde el nodo
    char* c;
```

```
    // Será cierto si el camino desde la raíz hasta este nodo contiene una palabra almacenada en el trie
    bool pal;
```

```
    // Puntero al siguiente nivel del trie
    TNodeTrie* sigLetra;
```

```
    // Puntero al siguiente nodo de la lista
    TListNode* sig;
```

```
};
```

```
class TLista {
```

```
public:
```

```
    // Constructor por defecto
    TLista();
```

```
    // Constructor de copia
    TLista(const TLista &);
```

```

        // Destructor
        ~TLista();

        // Sobrecarga del operador asignación
        TLista & operator=(const TLista &);
private:
        // Puntero al primer nodo de la lista
        TListaNodo* primero;
};

class TNodeTrie {
public:
        // Constructor por defecto
        TNodeTrie();

        // Constructor de copia
        TNodeTrie(const TNodeTrie &);

        // Destructor
        ~TNodeTrie();

        // Sobrecarga del operador asignación
        TNodeTrie & operator=(const TNodeTrie &);
private:
        // Lista con cada uno de las posibles letras de una palabra
        TLista listaCaracteres;
};

class TTrie {
public:
        // Constructor por defecto
        TTrie();

        // Constructor de copia
        TTrie(const TTrie &);

        // Destructor
        ~TTrie();

        // Sobrecarga del operador asignación
        TTrie & operator=(const TTrie &);
private:
        // Puntero a la raíz del trie
        TNodeTrie* raiz;
};

```

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED julio 2015

Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 4 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Es posible reconstruir un único árbol binario de búsqueda de n elementos ($n > 1$), a partir de su recorrido en inorden.	<input type="checkbox"/>	<input type="checkbox"/>	1	F
Sea un árbol binario lleno cuyo recorrido en inorden es: 10,15,17,20,21,28,35. La secuencia de árboles cuyas etiquetas son 35,28,20 es un camino en el mencionado árbol.	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Un multigrafo es un grafo que no tiene ninguna restricción: pueden existir arcos reflexivos y múltiples ocurrencias del mismo arco.	<input type="checkbox"/>	<input type="checkbox"/>	3	V
El número máximo de arcos que pueden existir en un grafo dirigido de n vértices son: $n(n-1) + n$.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
En un grafo dirigido, un ciclo es un camino simple en el que el vértice primero y último coinciden.	<input type="checkbox"/>	<input type="checkbox"/>	5	V
En la especificación algebraica, para definir la semántica de una operación de un tipo de datos sólo se pueden utilizar las operaciones generadoras constructoras.	<input type="checkbox"/>	<input type="checkbox"/>	6	F
El resultado del cálculo de la complejidad temporal en el mejor caso de un algoritmo X , da como resultado $n + n \cdot \log(n)$. Por lo tanto, diremos que la complejidad del algoritmo X cuando $n \rightarrow \infty$ pertenece a $\Omega(n)$.	<input type="checkbox"/>	<input type="checkbox"/>	7	F
El TAD vector visto en clase se define como un conjunto ordenado de pares <índice, valor>. Para cada índice definido dentro de un rango finito existe asociado un valor.	<input type="checkbox"/>	<input type="checkbox"/>	8	V
En un árbol AVL cuyo nodo raíz tiene un factor de equilibrio +1 siempre que se inserte un nuevo elemento hay que realizar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	9	F
Todo árbol completo es un árbol completamente equilibrado.	<input type="checkbox"/>	<input type="checkbox"/>	10	F
En un árbol 2-3, la altura siempre disminuye si la raíz es de tipo 2-nodo y al efectuar el borrado de un elemento es necesario realizar una combinación con el nodo raíz.	<input type="checkbox"/>	<input type="checkbox"/>	11	V
La operación de borrar un elemento en un árbol 2-3-4 finaliza cuando el nodo p es el nodo que contiene al elemento que se desea borrar.	<input type="checkbox"/>	<input type="checkbox"/>	12	F
La complejidad en su caso peor, de la unión de dos conjuntos implementados como listas no ordenadas de tamaño " n " y " m " respectivamente es de $O(n \cdot m)$.	<input type="checkbox"/>	<input type="checkbox"/>	13	V
Cuando implementamos un TAD Tabla de dispersión cerrada se usa una función de dispersión H tal que $H(x)$ devolverá un valor comprendido entre 0 y B , siendo B el número finito de clases en las que dividimos el conjunto.	<input type="checkbox"/>	<input type="checkbox"/>	14	F
El montículo mínimo o HEAP mínimo es un árbol binario completo que además es árbol mínimo.	<input type="checkbox"/>	<input type="checkbox"/>	15	V

Examen PED julio 2015

- Normas:**
- ♦ Tiempo para efectuar el examen: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Cada pregunta vale 2 puntos (sobre 10).**
 - Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.

1. Dada la sintaxis y la semántica de la operación Examen:

Examen: lista \rightarrow lista

Var L: lista; X,Y,a,b,c: Item;

Examen(crear()) = crear()

Examen(IC(crear(), X)) = IC(crear(), X)

Examen(IC(IC(L, Y), X)) = IC(IC(Examen(L, X), Y)

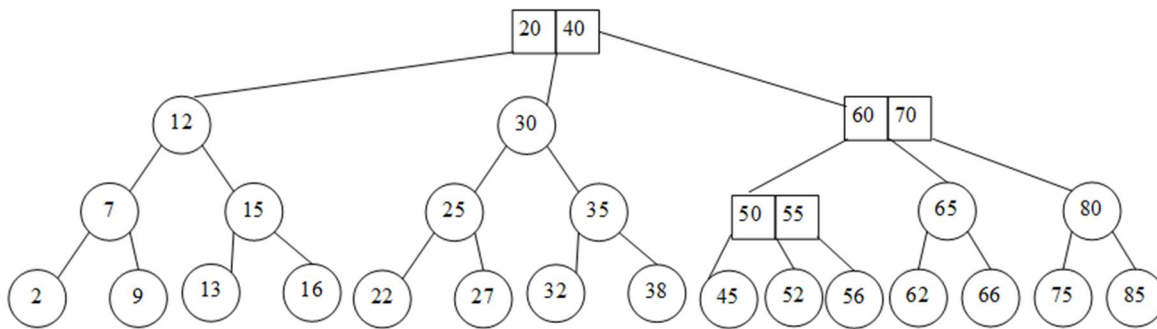
a) Explicar el funcionamiento de la operación Examen.

b) Aplicar la operación Examen a la lista: IC(IC(IC(crear(), c), b), a). Mostrar todos los pasos.

c) Sea una entidad bancaria que dispone de dos tipos de datos: (i) una lista (lista_de_clientes_morosos) que contiene los identificadores (números enteros) de los clientes morosos; (ii) un vector (vector_de_saldos) que almacena en cada posición (identificador del cliente) el saldo de todas las cuentas bancarias del cliente en la mencionada entidad bancaria.

Utilizando exclusivamente las operaciones generadoras constructoras (vistas en clase) del tipo lista y vector, definir la sintaxis y semántica de la operación “moroso” que actualiza el vector_de_saldos, asignando a cada cliente de la entidad bancaria que es moroso un saldo de “0”.

2. Sea el siguiente árbol A:



a) Sobre A, borrar el elemento 40 considerando A como un árbol 2-3 y como un árbol 2-3-4. (Criterios: (1) si el nodo tiene dos hijos hay que **SUSTITUIR POR el mayor de la izquierda**, (2) si el 2-nodo tiene dos hermanos, consultar el **hermano de la DERECHA**).

NOTA: Si no se realiza 1 gráfico para el borrado de cada elemento indicando las transformaciones realizadas, o bien no se siguen los criterios del enunciado: **NO SE VALORARÁ LA RESPUESTA**

b) Escribe un árbol 2-3 de altura 4, donde todos los elementos son enteros, en el que al insertar el elemento con etiqueta 10, la altura del árbol crezca.

3. Dadas las siguientes especificaciones:

- Crear una estructura vacía, con coste constante
- Consultar el máximo de todos los elementos, con coste constante
- Consultar el mínimo de todos los elementos, con coste constante
- Borrar el máximo, con una complejidad temporal (en su peor caso) logarítmica
- Borrar el mínimo, con una complejidad temporal (en su peor caso) logarítmica
- Insertar un elemento, con una complejidad temporal (en su peor caso) logarítmica

donde n es el número de elementos de la estructura sobre la cual tiene lugar la acción.

a) Indicar si existe algún tipo de datos (visto en clase) que cumpla todas las especificaciones. ¿Cuál sería? Pon un ejemplo para cada acción usando 10 elementos y justificando el coste temporal con una explicación.

b) Ordenar el siguiente vector de menor a mayor usando el algoritmo heapsort, indicando paso a paso las operaciones realizadas:

20	10	30	15	25	40	45	5
----	----	----	----	----	----	----	---

Hay que mostrar en cada operación realizada las transformaciones producidas en el vector.

Examen PED julio 2015. Soluciones

1. a) La operación Examen actúa sobre una lista y devuelve una lista. Los elementos que estén en posiciones pares (x) de la lista pasan a ocupar una posición impar anterior (x-1). Los que estén en posiciones impares (y) pasan a ocupar la posición par siguiente (y+1).

Si la lista contiene un número impar de elementos, el último no cambia su posición.

b) IC(IC(IC(crear(), c), a), b)

c) Según la definición de la operación asignar del tipo vector vista en clase:

si ($i < j$) entonces

asig(asig(v, i, x), j, y) = asig(asig(v, j, y), i, x)

si no asig(asig(v, i, x), j, y) = asig(v, i, y) **fsi**

Con lo que:

moroso: lista, vector \rightarrow vector

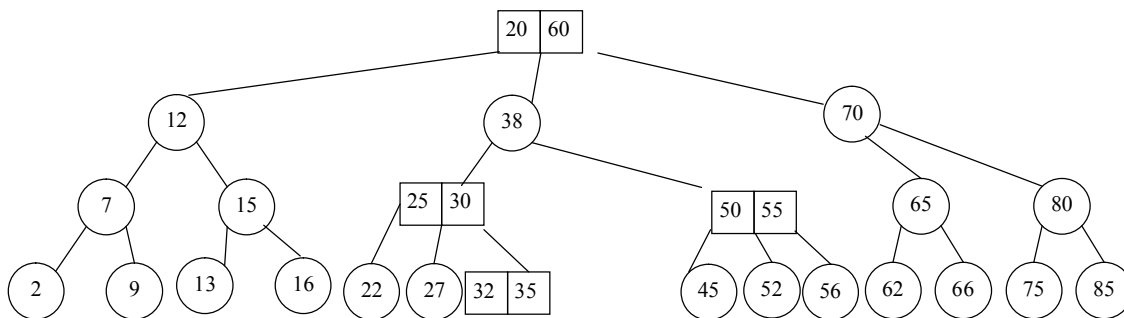
Var L: lista; V: vector; id: entero;

moroso(crearLista(), V) = V

moroso(L, crearVector()) = crearVector()

moroso(IC(L, id), V) = moroso(L, asig(V, id, 0)

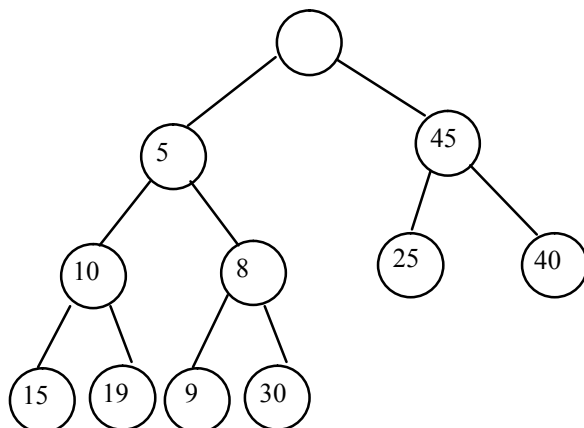
2. BORRADO DEL 40 COMO ÁRBOL 2-3: 2 COMBINACIONES Y 1 ROTACIÓN



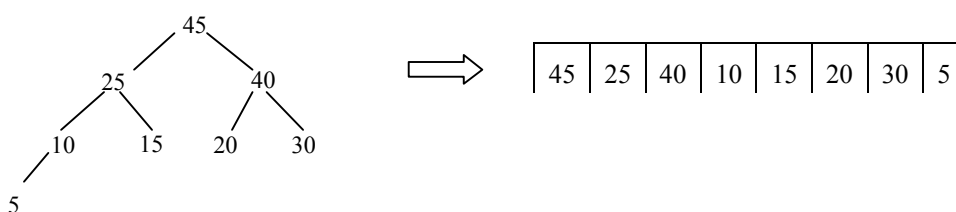
BORRADO DEL 40 COMO ÁRBOL 2-3-4: 2 ROTACIONES Y 1 COMBINACIÓN

3.

a) Un DEAP cumpliría todas las especificaciones

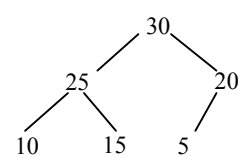


b) Primero se crea un heap máximo usando los elementos del vector inicial:

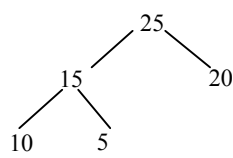


Después se borran todos los elementos del heap para conseguir el vector ordenado de menor a mayor.

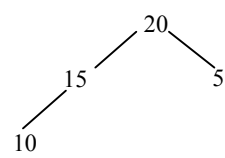
30	25	20	10	15	5	40	45
----	----	----	----	----	---	----	----



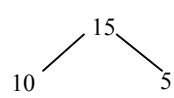
25	15	20	10	5	30	40	45
----	----	----	----	---	----	----	----



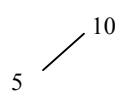
20	15	5	10	25	30	40	45
----	----	---	----	----	----	----	----



15	10	5	20	25	30	40	45
----	----	---	----	----	----	----	----



10	5	15	20	25	30	40	45
----	---	----	----	----	----	----	----



5	10	15	20	25	30	40	45
---	----	----	----	----	----	----	----

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED julio 2016

Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 4 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F							
La complejidad temporal (en su caso mejor) del siguiente fragmento de código es $\Omega(n)$ int i, length, n, i1, i2, k; for (i = 0, length = 1; i < n-1; i++) { for (i1 = i2 = k = i; k < n-1 && a[k] < a[k+1]; k++, i2++); if (length < i2 - i1 + 1) length = i2 - i1 + 1; }	<input type="checkbox"/>	<input type="checkbox"/>	1	V					
La complejidad temporal (en su peor caso) de la operación de insertar un elemento en una cola circular enlazada que no admite elementos repetidos es $O(n)$, siendo n el número de elementos de la cola.	<input type="checkbox"/>	<input type="checkbox"/>	2	V					
Un árbol con un único nodo es un árbol completo.	<input type="checkbox"/>	<input type="checkbox"/>	3	V					
El nivel de la raíz en un árbol binario es 0.	<input type="checkbox"/>	<input type="checkbox"/>	4	F					
Todo árbol binario mínimo es un árbol binario de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	5	F					
Un árbol binario de búsqueda completo es un AVL.	<input type="checkbox"/>	<input type="checkbox"/>	6	V					
El número de rotaciones que se nos pueden dar en el borrado de un elemento en un AVL son como máximo 3 menos que la altura del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	7	F					
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 en el mencionado árbol es $O(\log_2 n)$.	<input type="checkbox"/>	<input type="checkbox"/>	8	V					
En un árbol 2-3-4 los nodos pueden tener 1, 2, 3 ó 4 hijos.	<input type="checkbox"/>	<input type="checkbox"/>	9	F					
La mejor representación de los conjuntos siempre es el vector de bits porque es la más eficiente espacialmente.	<input type="checkbox"/>	<input type="checkbox"/>	10	F					
Sea una tabla de dispersión cerrada con estrategia de redispersión $h_i(x) = (H(x) + C \cdot i) \text{ MOD } B$, con $B=1000$ y $C=74$. Para cualquier clave "x" que se desee insertar, se recorrerán todas las posiciones de la tabla buscando una posición libre.	<input type="checkbox"/>	<input type="checkbox"/>	11	F					
El siguiente vector representa un montículo máximo: <table><tr><td>10</td><td>5</td><td>3</td><td>1</td><td>2</td></tr></table>	10	5	3	1	2	<input type="checkbox"/>	<input type="checkbox"/>	12	V
10	5	3	1	2					
Sea $G=(V,A)$ un grafo dirigido. Diremos que $G''=(V'',A'')$ es un árbol extendido de $G \Leftrightarrow V''=V, A'' \subset A, \forall v \in V'' \Rightarrow \text{grado}_E(v) \leq 1$	<input type="checkbox"/>	<input type="checkbox"/>	13	V					
Un digrafo es un multigrafo que no contiene arcos reflexivos.	<input type="checkbox"/>	<input type="checkbox"/>	14	F					
La especificación algebraica de la operación <i>longitud</i> definida en clase para el tipo lista es la siguiente: VAR L1: lista; x: item; <i>longitud</i> (crear()) = 0 <i>longitud</i> (inscabeza(L1, x)) = 1 + <i>inscabeza</i> (<i>longitud</i> (L1), x)	<input type="checkbox"/>	<input type="checkbox"/>	15	F					
En la especificación algebraica de un tipo de datos las operaciones modificadoras devuelven un valor de un tipo diferente al que se está definiendo.	<input type="checkbox"/>	<input type="checkbox"/>	16	F					

Examen PED julio 2016

- Normas:**
- ♦ Tiempo para efectuar el examen: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Cada pregunta vale 2 puntos (sobre 10).**
 - Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.

1. a) Utilizando exclusivamente las operaciones constructoras generadoras del **tipo pila**, definir la sintaxis y la semántica de la operación QuitaPares que actúa sobre una pila y devuelve una pila en la que se han eliminado las posiciones pares de la misma.

Nota: se asume que la posición 1 (impar) de la pila está en la cima de la misma.

b) Explicar las dos representaciones enlazadas del **tipo cola** vistas en clase definiendo los elementos que aparecen en la misma. Para cada una de ellas, explicar razonadamente (justificando la respuesta) la complejidad temporal (en su mejor y peor caso) de las operaciones encolar y desencolar.

2.

- a) Inserta en un árbol 2-3 inicialmente vacío los siguientes elementos: 10, 25, 20, 50, 60, 15, 30, 80
- b) ¿El árbol resultado obtenido en el apartado a) cumple las propiedades de árbol 2-3-4? Justifica tu respuesta. En caso afirmativo, suponiendo que el árbol resultado del apartado a) sea un 2-3-4, inserta los siguientes elementos: 35, 40, 45. En caso negativo, será un 2-3 y se insertarán los siguientes elementos: 70, 75, 5
- c) Sobre el resultado del apartado b) borra los elementos 45, 25 e indica de qué tipo es el árbol resultado. Criterios: Si el nodo tiene dos hijos substituir por el mayor de la izquierda. Si se realiza el borrado sobre un 2-3-4, en caso de tener dos nodos adyacentes a q entonces r será el hermano de la derecha. Si se realiza el borrado sobre un 2-3, en caso de tener dos hermanos consultar el hermano de la derecha.

3.

a) Escribir en C++ el código de una función para ordenar un vector de enteros (TVectorEnteros) en orden ascendente o descendente mediante un montículo doble (TDeap). Ejemplo de uso:

TVectorEnteros a; Ordenar(a, true); // Ordenación de menor a mayor

NOTAS: los errores de sintaxis de C++ se puntuarán de forma negativa; no hace falta definir el código de los métodos de las clases TDeap y TVectorEnteros, pero sí que habrá que definir los prototipos de los métodos que se utilicen (parámetros de entrada y salida); no es necesario que se hagan todas las operaciones sobre el mismo vector a ordenar.

b) Indicar RAZONADAMENTE su complejidad en el caso peor.

c) Ordenar el siguiente vector de menor a mayor usando dicha función, explicando las operaciones realizadas:

20	10	30	15	25	40	45	5	3	47	27	32	2	7	50	1
----	----	----	----	----	----	----	---	---	----	----	----	---	---	----	---

Examen PED julio 2016. Soluciones

1. a) QuitaPares: pila \rightarrow pila

Var p: pila; x,y: item;

QuitaPares (crear_pila())= crear_pila()

QuitaPares (apilar(crear_pila(), x))= apilar(crear_pila(), x)

QuitaPares (apilar(apilar(p, x), y))= apilar(QuitaPares(p), y)

b) Para la representación enlazada de las colas se utilizan punteros a **nodo**. El nodo contiene el **dato** a almacenar y un **puntero al siguiente nodo**. Se definen dos punteros adicionales: **tope** y **fondo**. **tope** apunta al primer elemento que hay que desencolar y **fondo** apunta al último elemento de la cola.

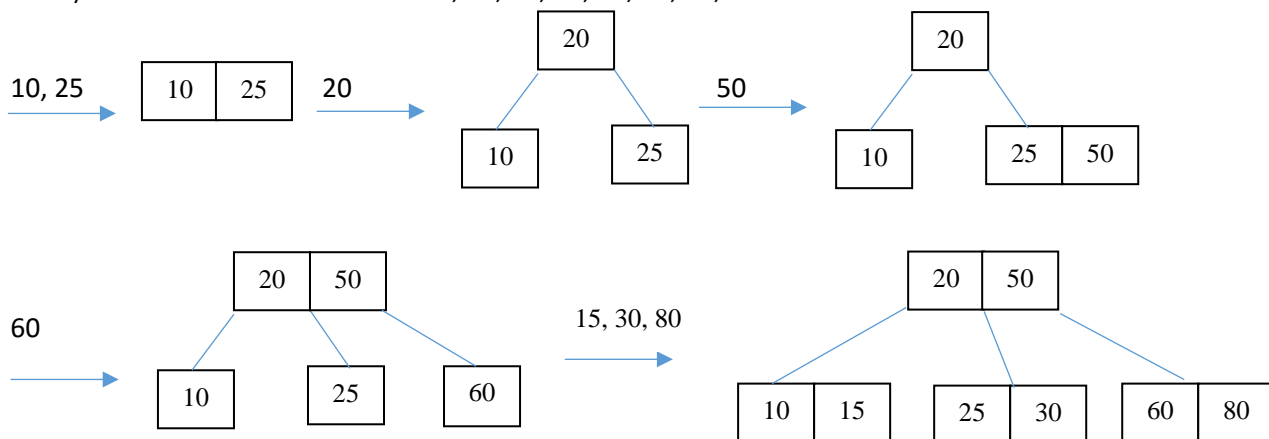
Utilizando esta estructura la complejidad temporal de las operaciones **encolar y desencolar** es la misma (ya que tenemos punteros a la primera y a la última posición de la cola): $\Omega(1) = O(1) = \theta(1)$

Esta representación se optimiza con las **colas circulares enlazadas**, en las que sólo se necesita un puntero (**fondo**) al último elemento de la cola; el siguiente elemento de fondo apunta al primer elemento de la cola.

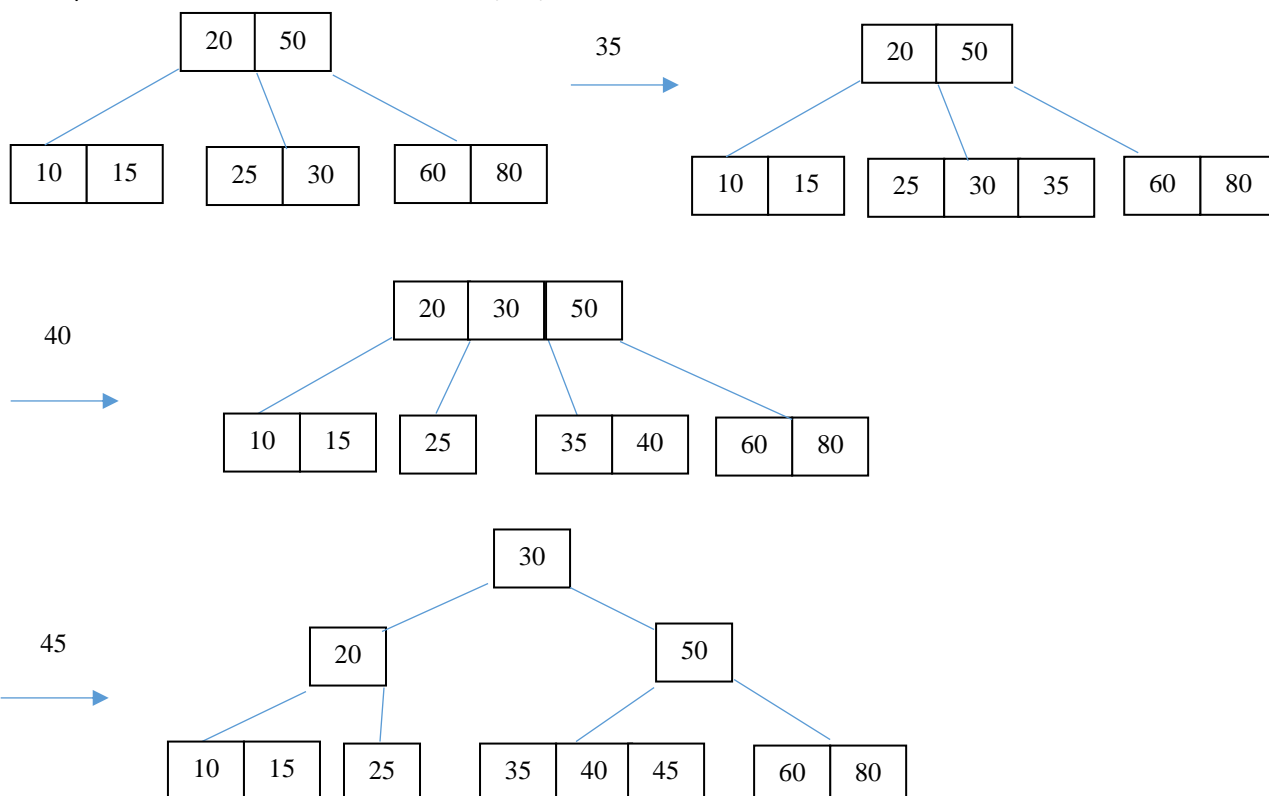
La complejidad temporal de las operaciones **encolar y desencolar** es la misma (ya que tenemos un puntero que apunta a la última posición y al primero –siguiente de fondo–): $\Omega(1) = O(1) = \theta(1)$

2.

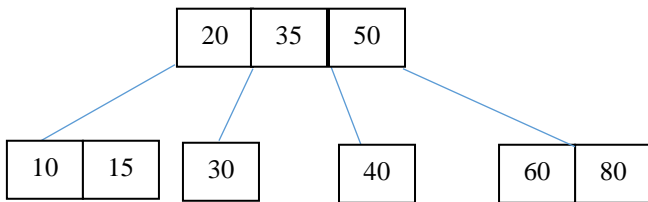
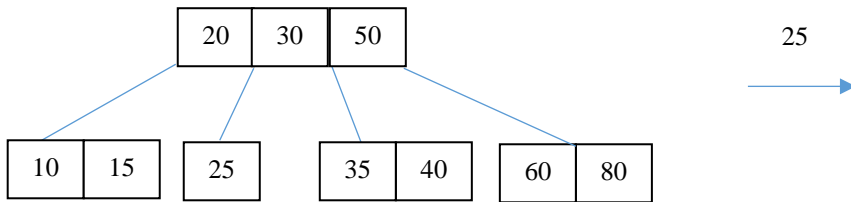
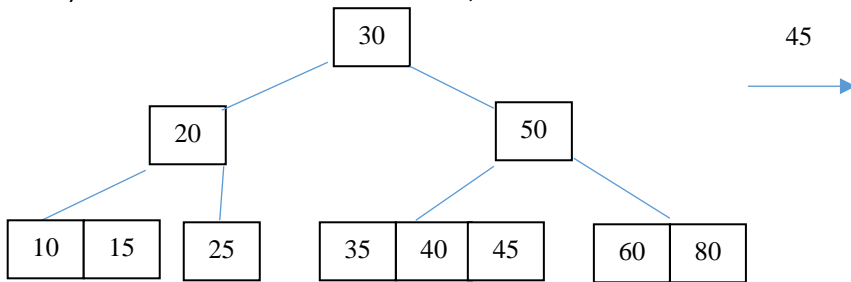
a) Insertar en un árbol 2-3: 10, 25, 20, 50, 60, 15, 30, 80



b) Insertar en un árbol 2-3-4: 35, 40, 45



c) Borrar en un árbol 2-3-4: 45,25



3.

a)

```

class TDeap {
public:
    void Insertar(int);           // Inserta un elemento
    int Maximo(void);             // Devuelve el máximo del Deap
    int Minimo(void);             // Devuelve el mínimo del Deap
    void BorrarMaximo(void);      // Borra el máximo del Deap
    void BorrarMinimo(void);      // Borra el mínimo del Deap
    ...
};

class TVectorEnteros {
public:
    int Tamanyo(void);           // Devuelve el número de elementos del vector
    ...
};

void Ordenar(TVectorEnteros& a, const bool& menorAmayor) {
    TDeap d; int i;

    for(int i = 1; i <= d.Tamanyo(); ++i) {
        d.Insertar(a[i]);
    }
    if(menorAmayor) {
        for(i = 1; i <= d.Tamanyo(); ++i) {
            a[i] = d.Minimo();
            d.BorrarMinimo();
        }
    }
}
    
```

```

else {
    for(i = 1; i <= d.Tamanyo(); ++i) {
        a[i] = d.Maximo();
        d.BorrarMaximo();
    }
}

```

b) Tendría una complejidad $O(n \log n)$, con n el número de elementos del vector, al igual que el algoritmo HeapSort que utiliza un montículo simple, ya que igualmente habría que realizar n operaciones de inserción y borrado, cada una de ellas con una complejidad $O(\log n)$.

Apellidos:

Nombre:

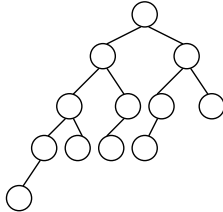
Convocatoria:

DNI:

Examen TAD/PED julio 2010

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - El test vale un 40% de la nota de teoría: 4 puntos.
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: $F: \text{natural} \rightarrow \text{BOOL}$, y su semántica como: $F(\text{cero}) = \text{TRUE}$, $F(\text{suc}(\text{cero})) = \text{FALSE}$, $F(\text{suc}(\text{suc}(x))) = F(x)$. Para el número natural $x=35$, la función F devolvería FALSE.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
En C++, la memoria que se reserva con new se libera automáticamente por el destructor.	<input type="checkbox"/>	<input type="checkbox"/>	2	F
La mejor complejidad temporal que se puede conseguir en un algoritmo es $O(n)$, con "n" como la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
La semántica de la operación <i>sublista</i> del tipo lista vista en clase es la siguiente: $\text{VAR } L: \text{lista}; x, y: \text{item}; n: \text{natural}; p: \text{posicion};$ $\text{sublista}(L, p, 0) = \text{crear}()$ $\text{sublista}(\text{crear}(), p, n) = \text{crear}()$ $\text{si } p == \text{primera}(\text{inscabeza}(L, x)) \text{ entonces}$ $\quad \text{sublista}(\text{inscabeza}(L, x), p, n) = \text{inscabeza}(\text{sublista}(L, \text{primera}(L), n), x)$ $\text{si no sublista}(\text{inscabeza}(L, x), p, n) = \text{sublista}(L, p, n)$	<input type="checkbox"/>	<input type="checkbox"/>	4	F
Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol	<input type="checkbox"/>	<input type="checkbox"/>	5	V
Cuando realizamos un recorrido en inorden en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor	<input type="checkbox"/>	<input type="checkbox"/>	6	V
Todo árbol binario de búsqueda es un árbol mínimo	<input type="checkbox"/>	<input type="checkbox"/>	7	F
El siguiente árbol está balanceado con respecto a la altura 	<input type="checkbox"/>	<input type="checkbox"/>	8	V
Todo árbol binario de búsqueda es un árbol 2-3.	<input type="checkbox"/>	<input type="checkbox"/>	9	F
En un árbol 2-3-4 sólo los nodos hoja y la raíz pueden ser de tipo 2-nodo	<input type="checkbox"/>	<input type="checkbox"/>	10	F
En un árbol rojo-negro la búsqueda de una etiqueta dependerá de los colores de los hijos de cada nodo	<input type="checkbox"/>	<input type="checkbox"/>	11	F
El árbol 2-3 es un árbol B m-camino de búsqueda con $m=2$	<input type="checkbox"/>	<input type="checkbox"/>	12	F
El TAD Diccionario es un subtipo del TAD Conjunto	<input type="checkbox"/>	<input type="checkbox"/>	13	V
La dispersión abierta elimina el problema del clustering secundario.	<input type="checkbox"/>	<input type="checkbox"/>	14	V
En un montículo doble de altura h se pueden almacenar un máximo de 2^{h-2} claves.	<input type="checkbox"/>	<input type="checkbox"/>	15	F
Un árbol leftist mínimo es un árbol binario mínimo tal que si no es vacío: $\text{CMÍN}(\text{HijoIzq}(x)) > \text{CMÍN}(\text{HijoDer}(x))$ para todo x no vacío	<input type="checkbox"/>	<input type="checkbox"/>	16	F

Examen PED julio 2010

- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** Se publicará en el campus virtual.

• **Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas**

1. Realizar la especificación algebraica de la función $ABB(arbin) \rightarrow Bool$, la cual determina si el árbol binario que recibe como parámetro, es un árbol binario de búsqueda.

Notas:

- Un árbol binario es un árbol binario de búsqueda si al realizar el recorrido en inorden todos los elementos del árbol aparecen ordenados de menor a mayor.
- Se supone que está definida la siguiente operación: $inorden(arbin) \rightarrow Lista$.
- Se pueden utilizar funciones definidas en clase para una lista. Habrá que escribir la especificación algebraica del resto de funciones auxiliares que se utilicen.

2. Sea el siguiente recorrido PREORDEN de un árbol AVL: 10, 5, 3, 1, 4, 7, 16, 13, 11, 12, 14, 19, 21

- Reconstruye el árbol resultante a partir del recorrido Preorden anterior.
- En el árbol AVL resultante del apartado a), inserta los siguientes elementos: 20, 18, 17.
- Del árbol resultante del apartado B), borra los elementos 7, 21, 19.

Nota: Utiliza los criterios vistos en clase.

3.

- Construir un árbol de búsqueda digital con los elementos del 1 al 10 insertados en ese orden (1, 2, 3...10).
- Sobre el árbol de búsqueda digital anterior, calcular la complejidad temporal, en el peor de los casos, de las operaciones de inserción y búsqueda; dicha complejidad debe estar en función de la clave. Justifica la respuesta (en caso contrario, no se evaluará este apartado del ejercicio).

4. Dado el grafo dirigido representado por la lista de de adyacencia que se muestra a continuación:

- Obtened el árbol extendido en profundidad partiendo del vértice 1 y la clasificación de los arcos. Para la clasificación, los arcos se etiquetarán en la misma lista de adyacencia (**de otro modo no se corregirán**), por ejemplo:

A C
1 → 3 → 5

- Realizad el recorrido en anchura partiendo del vértice 1.
- Indica el menor número posible de arcos que habría eliminar para convertirlo en un grafo acíclico dirigido.
- Indica el menor número posible de arcos que habría eliminar para convertirlo en un árbol.

Nota: La lista de adyacencia de cada vértice se recorre en el mismo orden en que aparecen en la lista para todos los casos del ejercicio. Para vértices no visitados, continuar por el menor vértice.

1 → 3 → 5 → 8 → 10 → 11 → 12
2 → 1 → 4 → 8 → 10 → 11 → 12
3 → 2 → 4 → 10 → 11 → 12
4 → 5 → 1 → 8 → 10 → 11 → 12
5 → 2 → 3 → 10 → 11 → 12
6 → 9
7 → 6 → 9 → 5
8 → 5 → 3 → 10 → 11 → 12
9 → 6
10 →
11 →
12 → 11

Examen PED julio 2010. Soluciones

1.

ListaOrdenada(Lista) \rightarrow Bool

inorden(arbin) \rightarrow Lista

ABB(arbin) \rightarrow Bool

VAR x: item; L:Lista; i,d: Arbin;

ListaOrdenada(crear()) = TRUE

ListaOrdenada(incabeza(crear(), x) = TRUE

ListaOrdenada(incabeza(L,x)) =

Si (x < obtener(L, primera(L))

ListaOrdenada(L)

Sino

FALSE

Fsi

ABB(crear()) = TRUE

ABB(enraizar(crear(), x, crear()) = TRUE

ABB(enraizar(i, x, d))=

si ListaOrdenada(inorden(enraizar(i, x, d)))

entones

TRUE

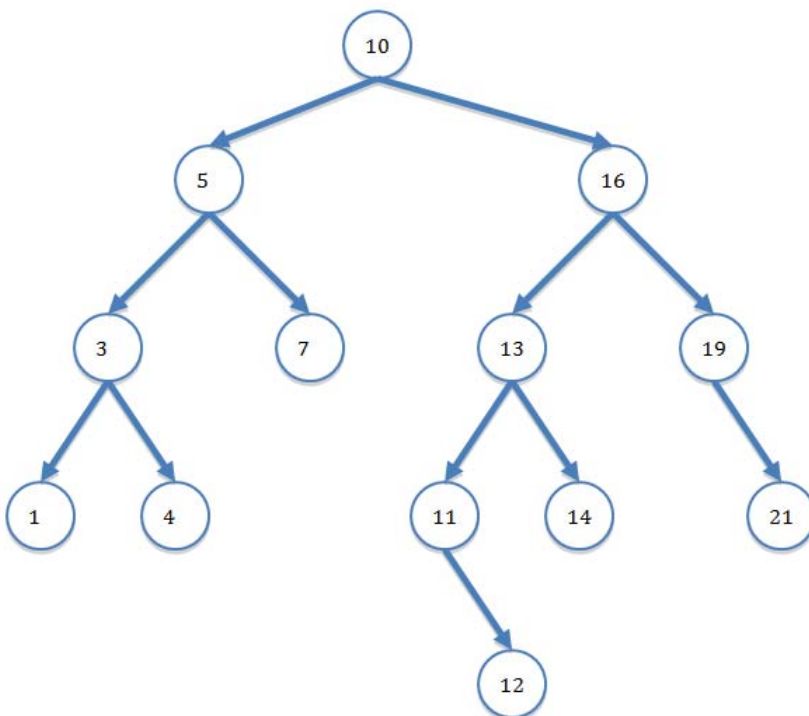
sino

FALSE

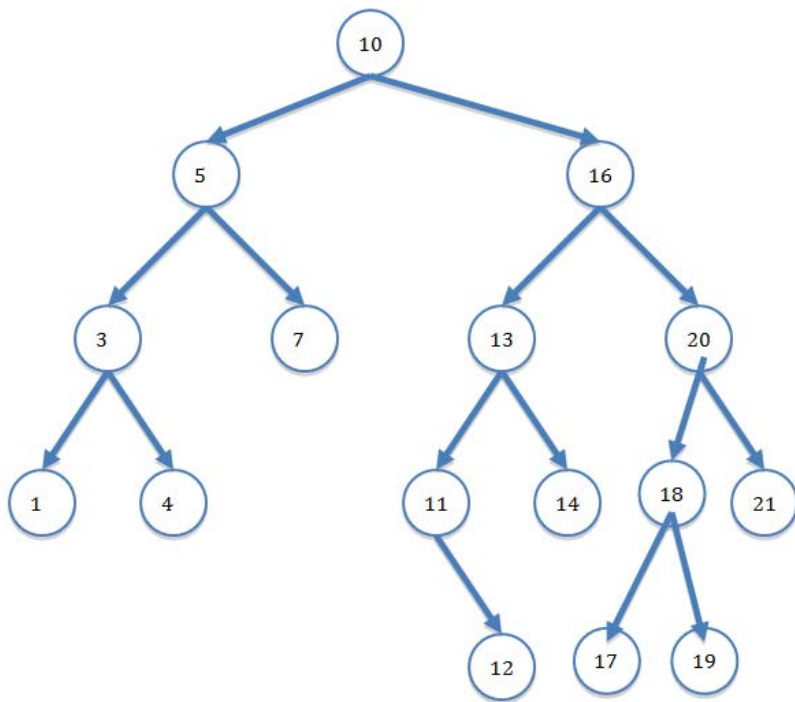
fsi

2.

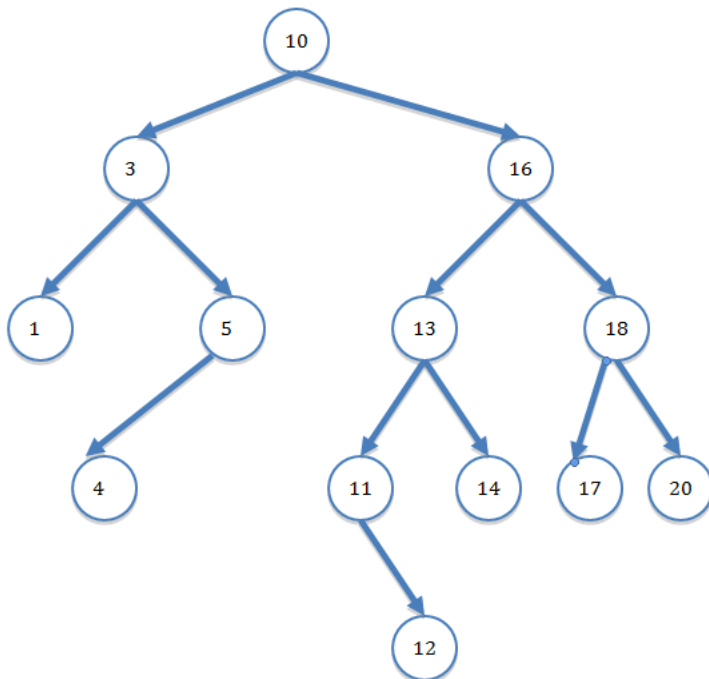
a)



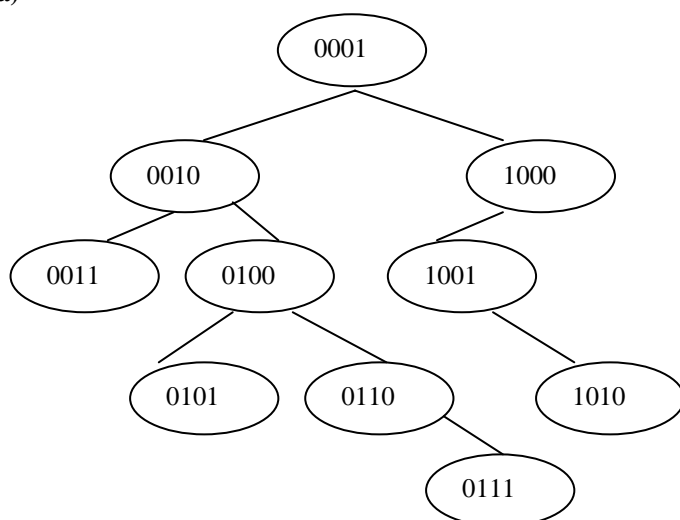
b)



c)



3. a)



- b)** Para la inserción y búsqueda $\rightarrow O(N+1)$ siendo N el número de bits de la clave, es decir $O(5)$.
 En el peor de los casos tenemos que recorrer la altura del árbol, que será el nº de bits de la clave más 1.
 Con 4 bits podemos representar $2^4 = 16$ elementos; para representar 16 elementos en un árbol binario, éste debe tener altura 5 (un árbol binario con altura h tiene como máximo $2^h - 1$ nodos; con altura 4 tendrá como máximo $2^4 - 1 = 15$ nodos; con altura 5 tendrá como máximo $2^5 - 1 = 31$ nodos).

4.

- a)** DFS = 1, 3, 2, 4, 5, 10, 11, 12, 8, 6, 9, 7

```

      A   AV AV  AV  AV  AV
1  → 3 → 5 → 8 → 10 → 11 → 12
      R   A   AV AV  AV  AV
2  → 1 → 4 → 8 → 10 → 11 → 12
      A   AV AV  AV  AV
3  → 2 → 4 → 10 → 11 → 12
      A   R   A  AV  AV  AV
4  → 5 → 1 → 8 → 10 → 11 → 12
      R   R   A   A   A
5  → 2 → 3 → 10 → 11 → 12
      A
6  → 9
      C   C   C
7  → 6 → 9 → 5
      C   R   C   C   C
8  → 5 → 3 → 10 → 11 → 12
      R
9  → 6

10 →

11 →
      C
12 → 11

```

- b)** BFS = 1, 3, 5, 8, 10, 11, 12, 2, 4
- c)** Los arcos de retroceso
- d)** Los arcos de retroceso, de avance y de cruce

Apellidos:

Nombre:

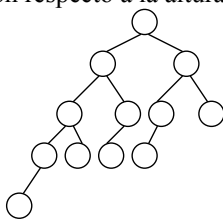
Convocatoria:

DNI:

Examen PED julio 2011

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En C++, la expresión <code>return &c;</code> devuelve la dirección de memoria de la variable <code>c</code> .	<input type="checkbox"/>	<input type="checkbox"/>	1	V
En C++, una función no puede tener todos sus parámetros con valores por omisión o por defecto.	<input type="checkbox"/>	<input type="checkbox"/>	2	F
En la escala de complejidades se cumple que $O(\log n) \subset O(\log \log n)$.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
La operación <code>BorrarItem</code> , que borra todas las ocurrencias del ítem <code>i</code> que se encuentren en la lista, tiene la siguiente sintaxis y semántica: 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)	<input type="checkbox"/>	<input type="checkbox"/>	4	V
Un árbol con un único nodo tiene un único camino cuya longitud es 1	<input type="checkbox"/>	<input type="checkbox"/>	5	F
En cualquier tipo de datos árbol, cada elemento puede tener varios predecesores, pero como máximo un sucesor.	<input type="checkbox"/>	<input type="checkbox"/>	6	F
El siguiente árbol está balanceado con respecto a la altura 	<input type="checkbox"/>	<input type="checkbox"/>	7	V
Si se inserta un elemento en un árbol 2-3 y todos los nodos que están en el camino desde la raíz a la hoja donde se inserta el elemento son del tipo 3-nodo, la altura del árbol 2-3 resultante crece con respecto al árbol 2-3 original.	<input type="checkbox"/>	<input type="checkbox"/>	8	V
En un árbol 2-3-4 de altura 3 donde todos sus nodos son del tipo 3-nodo, el número de elementos total es 27.	<input type="checkbox"/>	<input type="checkbox"/>	9	F
En un árbol rojo-negro, el número de enlaces negros ha de ser mayor que el de enlaces rojos.	<input type="checkbox"/>	<input type="checkbox"/>	10	F
El nodo de un árbol B m-camino de búsqueda con $m=100$ puede tener como máximo 99 claves.	<input type="checkbox"/>	<input type="checkbox"/>	11	V
La complejidad temporal, en su peor caso, de la operación de Unión entre 2 conjuntos con m elementos cada uno y representados con una lista ordenada es $O(m)$.	<input type="checkbox"/>	<input type="checkbox"/>	12	V
En el Hash cerrado la tabla de dispersión de tamaño B se tiene que reestructurar cuando se cumpla que el número de elementos $n \geq 2B$.	<input type="checkbox"/>	<input type="checkbox"/>	13	F
En un TRIE la complejidad temporal en su peor caso de la función Pertenece es $O(n)$ siendo n el número de nodos del árbol	<input type="checkbox"/>	<input type="checkbox"/>	14	F

Examen PED julio 2011

- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** Se publicará en el campus virtual.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Dar la sintaxis y la semántica de la operación **mezcla**, que actúa sobre dos colas y devuelve una cola nueva con los elementos de las dos colas encolados de forma alternada y en el mismo orden que aparecen en cada cola.

Por ejemplo, suponiendo que C1 y C2 son dos colas, escritas de izquierda a derecha desde la cabeza al fondo de la cola, la mezcla de C1 y C2 debería dar el siguiente resultado:

C1 = (a b c)

C2 = (x y)

$mezcla(C1, C2) = (a \ x \ b \ y \ c)$

2. Dados los siguientes recorridos de un árbol:

Preorden: 16, 12, 30, 21, 4, 8

Inorden: 12, 30, 16, 4, 8, 21

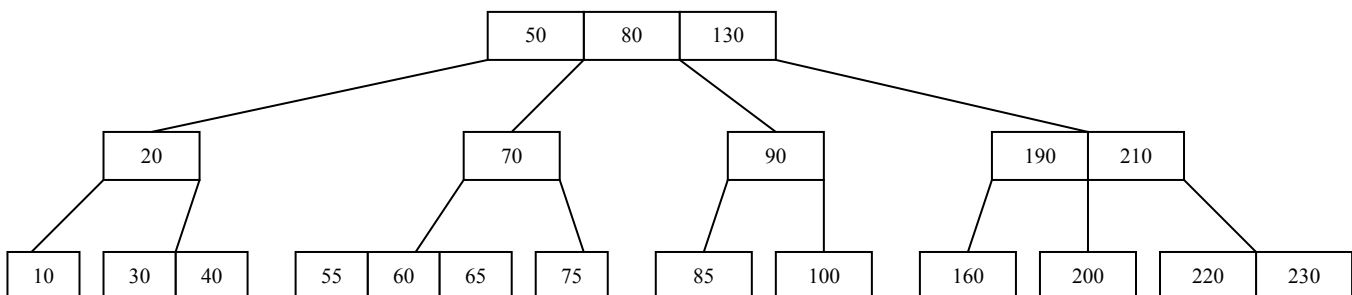
- Obtener el árbol binario correspondiente a estos recorridos.
- ¿Qué condiciones deben cumplir los elementos del recorrido Inorden para que el árbol resultante sea un árbol binario de búsqueda?
- Manteniendo los mismos elementos, realiza los cambios oportunos en los recorridos para que el árbol resultante cumpla las condiciones de ABB y AVL.
- Escribe el pseudocódigo del recorrido por niveles de un árbol binario.

3. Sea un montículo doble (DEAP) inicialmente vacío.

- Inserta los siguientes ítems en el montículo doble vacío: 12, 19, 10, 16, 18, 17, 30, 22, 3, 21, 1.
- Del montículo doble resultante del apartado a, extrae y borra el máximo y seguidamente el mínimo.
- Explica cual es la complejidad temporal en el peor caso de insertar un elemento en un montículo doble. Razona tu respuesta. Explica cual es la complejidad temporal en el peor caso de la operación Borrar (ítem), que borra del montículo doble el ítem que pasamos por parámetro.

4. Dado el siguiente árbol 2-3-4:

- Convertirlo a R-N. Para el caso de los 3-nodo, utilizar la conversión en la que el ítem izquierdo queda como hijo del ítem medio.
- Sobre el árbol R-N resultado, insertar sucesivamente y en este orden las claves 67, 69 y 68, detallando los cambios de color y rotaciones empleadas. No será válido, realizar la inserción como si fuese un árbol 2-3-4 y realizar una transformación final a R-N, por ello, habrá que detallar claramente los cambios de color y rotaciones empleadas, en caso contrario no se puntuará la pregunta.



Examen PED julio 2011. Soluciones

1.

SINTAXIS

mezcla cola, cola \rightarrow cola

Métodos auxiliares:

mezclaAux cola, cola, cola \rightarrow cola

invierte cola \rightarrow cola

invierteAux cola, cola \rightarrow cola

SEMÁNTICA

VAR c, d, e: cola; x, y: ítem;

mezcla(c, d) = mezclaAux(invierte(c), invierte(d), crear())

mezclaAux(crear(), crear(), c) = c

mezclaAux(crear(), encolar(c, x), d) = mezclaAux(crear(), c, encolar(d, x))

mezclaAux(encolar(c, x), crear(), d) = mezclaAux(c, crear(), encolar(d, x))

mezclaAux(encolar(c, x), encolar(d, y), e) = mezclaAux(c, d, encolar(encolar(e, x), y))

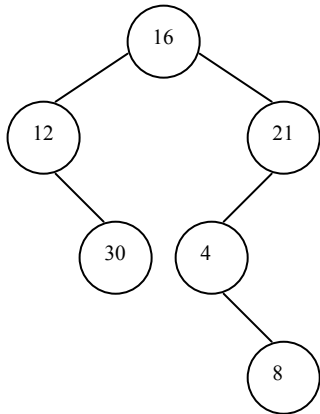
invierte(c) = invierteAux(c, crear())

invierteAux(crear(), c) = c

invierteAux(encolar(c, x), d) = invierteAux(c, encolar(d, x))

2.

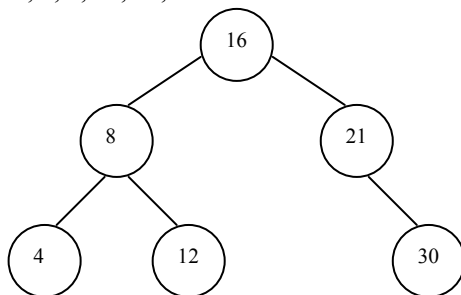
a)



b) Los elementos deben estar ordenados de menor a mayor

c) Inorden: 4, 8, 12, 16, 21, 30

Preorden: 16, 8, 4, 12, 21, 30



d)

algoritmo Niveles (a:arbin)

var c: cola de arbin; aux: arbin; fvar

Encolar(c, a)

mientras no EsVacia(c) hacer

 aux := Cabeza(c)

 Escribe(Raiz(aux))

 Desencolar(c)

 si no EsVacio(HijoIz(aux)) entonces Encolar(c, HijoIz(aux)) fsi

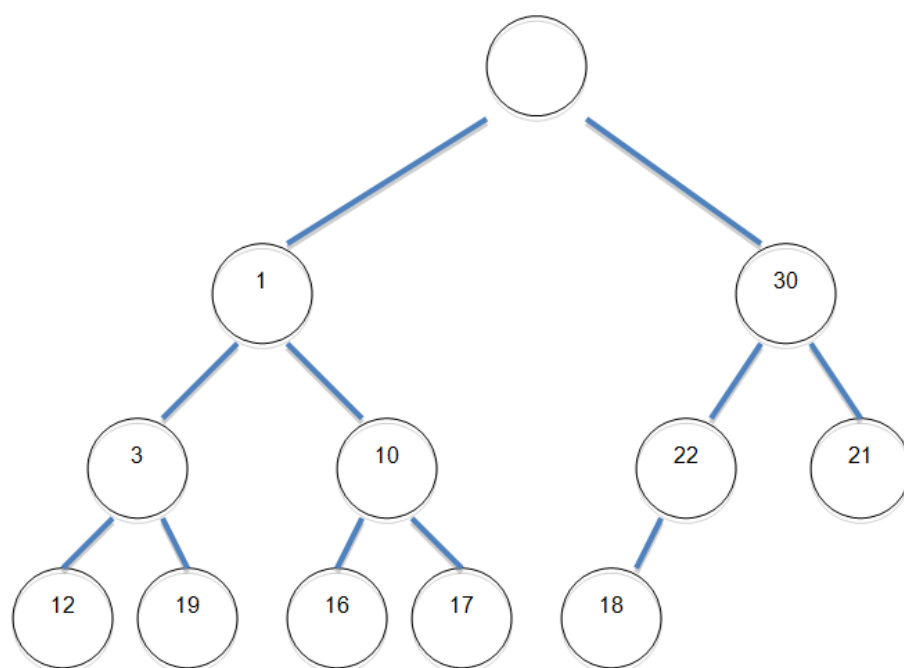
 si no EsVacio(HijoDe(aux)) entonces Encolar(c, HijoDe(aux)) fsi

fmientras

falgoritmo

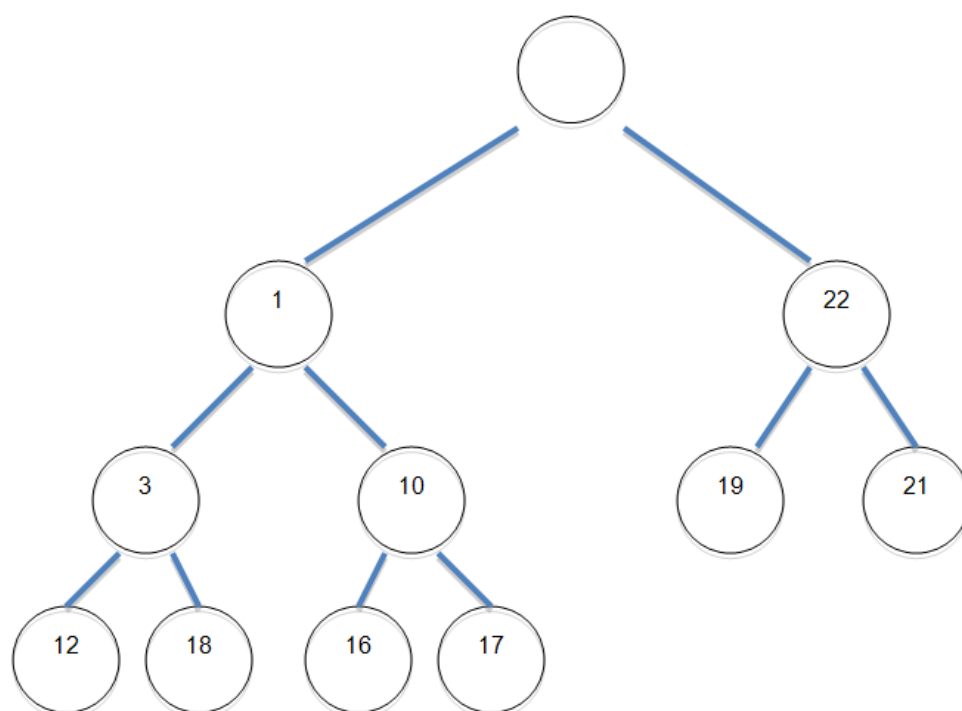
2.

a)

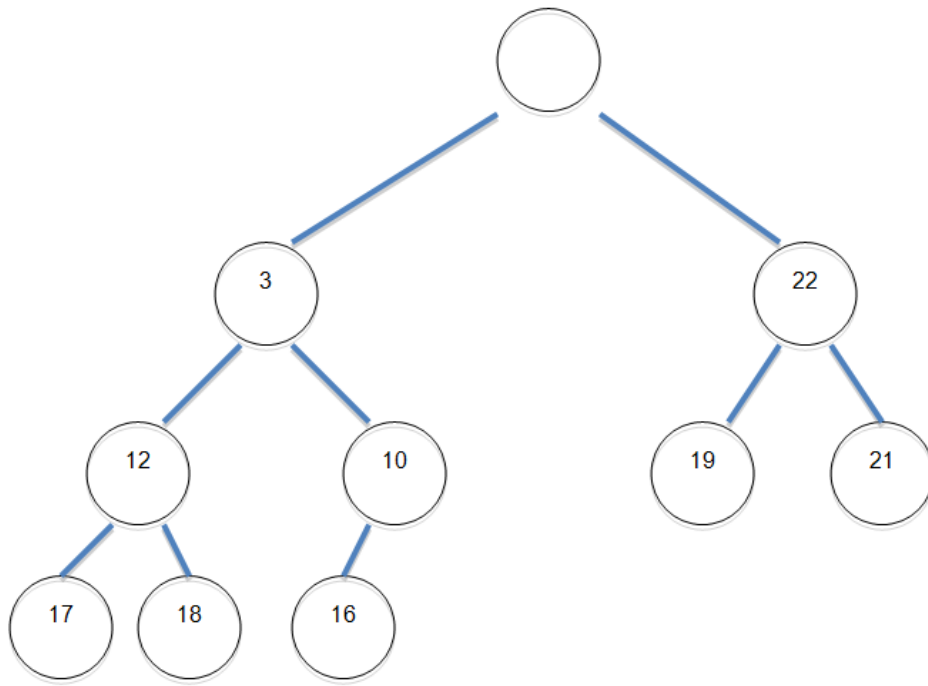


b)

Borrar el 30



Borrar el 1

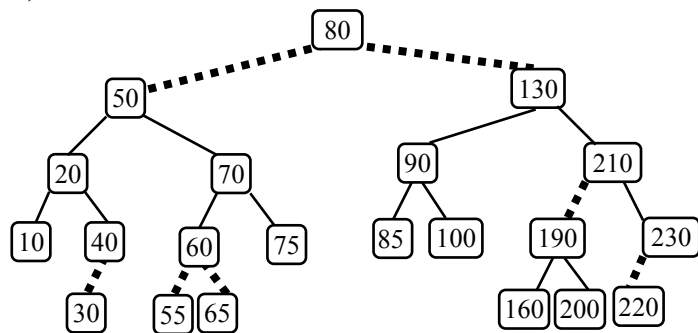


c) Al realizar una inserción, insertamos en el último elemento que sigue manteniendo la propiedad de árbol binario completo, por lo que estamos insertando en el último nivel del árbol. Para hacer las reestructuraciones necesarias nos vamos a mover hacia arriba y en el peor de los casos llegar hasta la raíz. Como es un árbol completamente equilibrado, el número de alturas que tenemos que recorrer en el peor caso son $h = \log_2 n$. Por lo tanto la complejidad es $O(\log_2 n)$

- La complejidad de la operación Borrar(ítem), en un montículo doble (DEAP) será en el peor caso $O(n)$, ya que no sabemos la posición que ocupa el ítem y es posible que haya que borrar todos los ítems del montículo. Si además quisiéramos mantener el resto de elementos en el montículo, es decir solamente borrar el elemento que pasamos por parámetro y no eliminar el resto, tendríamos que insertar en un montículo nuevo todos los elementos del montículo viejo, excepto el que queremos borrar. Esto tendría complejidad $O(n + \log n)$, en definitiva $O(n)$

4)

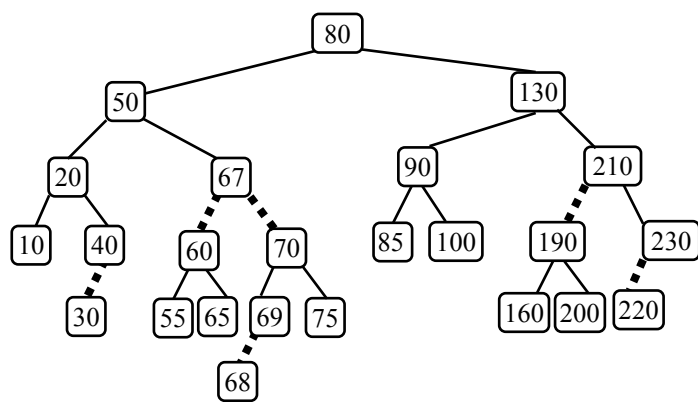
a)



b) Insertar 67: 2 cambios de color.

Insertar 69: Rotación II

Insertar 68: Cambio de color y Rotación ID



Apellidos:

Nombre:

Convocatoria:

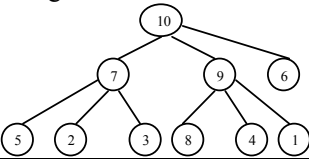
DNI:

Examen PED junio 2015

Modalidad 0

Normas:

- Tiempo para efectuar el test: **25 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 2 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Para el tratamiento de errores en la especificación algebraica, se añaden funciones constantes que devuelven un valor del tipo que causa el error.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
La complejidad temporal (en su caso peor) del siguiente fragmento de código es $O(n^2)$ <pre>int i, j, n, sum; for (i = 4; i < n; i++) { for (j = i-3; sum = a[i-4]; j <= i; j++) sum += a[j]; cout << "La suma del subarray " << i-4 << " es " << sum << endl; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Es posible obtener una representación enlazada de una cola utilizando un único puntero que apuntará al fondo de la cola.	<input type="checkbox"/>	<input type="checkbox"/>	3	V
Dado un único recorrido de cualquier árbol, siempre es posible reconstruir dicho árbol	<input type="checkbox"/>	<input type="checkbox"/>	4	F
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es logarítmica respecto a la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	5	F
La complejidad temporal en el peor caso y en el mejor caso de la operación inserción en un AVL son lineal y logarítmica respecto al número de nodos en el árbol	<input type="checkbox"/>	<input type="checkbox"/>	6	F
El borrado en un árbol AVL puede requerir una rotación en todos los nodos del camino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	7	V
Dado un árbol 2-3 de altura h con n items: $2^h - 1 \leq n \leq 3^h - 1$	<input type="checkbox"/>	<input type="checkbox"/>	8	V
Los nodos hoja de un árbol 2-3 han de estar en el mismo nivel del árbol	<input type="checkbox"/>	<input type="checkbox"/>	9	V
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	<input type="checkbox"/>	<input type="checkbox"/>	10	V
Un árbol 2-3-4 es un árbol 4-camino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	11	V
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (Operación(Conjunto) \rightarrow Natural; Var: C: Conjunto; x: Ítem): Operación(Crear) $\Leftrightarrow 1$ Operación (Insertar(C, x)) $\Leftrightarrow 3 + \text{Operación}(C)$	<input type="checkbox"/>	<input type="checkbox"/>	12	F
En un montículo el número de claves en el hijo izquierda de la raíz es mayor o igual que en su hijo derecha	<input type="checkbox"/>	<input type="checkbox"/>	13	V
El siguiente árbol es un montículo máximo: 	<input type="checkbox"/>	<input type="checkbox"/>	14	F
La siguiente secuencia de nodos de un grafo es un ciclo: 1,2,3,2,1	<input type="checkbox"/>	<input type="checkbox"/>	15	F
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de retroceso es un grafo acíclico dirigido.	<input type="checkbox"/>	<input type="checkbox"/>	16	F

Examen PED junio 2015

- Normas:**
- ♦ Tiempo para efectuar el examen: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Cada pregunta vale 2 puntos (sobre 10).**
 - **Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.**

1. Dado el grafo **no dirigido** representado por la lista de adyacencia que se muestra a continuación (se recorrerá cada lista de adyacencia considerándola ordenada de menor a mayor):

a) Obtener DFS(1), el árbol extendido en profundidad partiendo del vértice 1 y la clasificación de las aristas.

b) Obtener BFS(1).

1 → 2 → 4

2 → 9

3 → 1 → 7

5 → 6 → 3 → 1

6 → 1

7 → 1 → 5

8 → 7

9 → 1

10 → 14

11 → 10

12 → 11 → 13 → 10

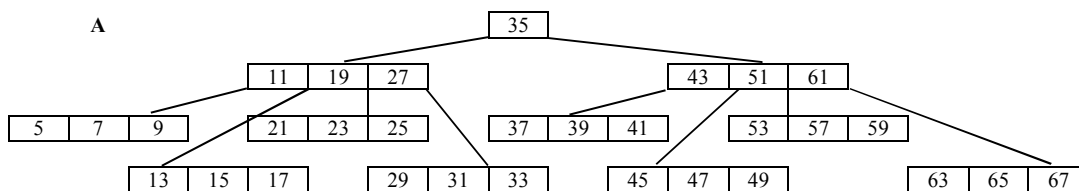
13 → 10 → 11

14 → 11 → 12 → 13

c) Utilizando exclusivamente las operaciones constructoras generadoras de grafo, definir la sintaxis y la semántica de la operación *CalculaPesos* que actúa sobre un grafo dirigido ponderado donde: los vértices son números Naturales y los Item son números Naturales que representan distancias kilométricas para cada par de vértices. *CalculaPesos* devuelve la suma de las distancias kilométricas de todos los arcos del grafo entre vértices pares.

Nota: se pueden utilizar todas las operaciones definidas para números naturales.

2. Sea el siguiente árbol 2-3-4:



a) Sobre este árbol A, realiza la inserción de los ítems 1, 24 y 58 en ese orden.

b) Escribe un árbol 2-3-4 de altura 4 en el que al borrar el ItemIzquierda del nodo raíz, haya que realizar 2 combinaciones y 1 rotación (da igual el orden en que se realicen) considerando los siguientes criterios. Realizar dicho borrado.

Criterio 1: r es el hermano de la izquierda.

Criterio 2: Si el ítem a borrar no está en una hoja, sustituir por el mayor de la izquierda.

3. Dada la siguiente función que calcula el código ASCII de una cadena de caracteres:

```
int hashChar(char *cadena, int max=20000){
    int cod=0;
    for (int i=0; i<strlen(cadena); i++)
        if (cod<max)
            cod=cod+toascii(cadena[i]);
        else
        {
            cod=cod%100;
            cod=cod+toascii(cadena[i]);
        }
    return(cod);
}
```

Códigos ASCII de cada letra:

A=65	C=67	E=69	J=74	M=77
N=78	O=79	P=80	R=82	S=83

a) Indicar la complejidad temporal. ¿Existe mejor y peor caso?

b) Si no se utilizara el parámetro max, es decir, que dentro del for sólo estuviera la instrucción *cod = cod + toascii(cadena[i]);* ¿Habría algún caso en que la función hashChar no funcionara correctamente?

c) Insertar en una tabla de dispersión cerrada de tamaño B=7, con función de dispersión $H(x) = x \text{ MOD } B$ y con estrategia de redispersión segunda función hash, la siguiente secuencia de elementos:

- 1) AMOR(303) – 2) COSA(294) – 3) MORA(303)
- 4) ESPONJA(528) – 5) RAMO(303) –
- 6) JAPONES(528) – 7) ROMA(303) –
- 8) CASO(294)

¿Qué elemento o elementos no se podrían insertar en la tabla? ¿Por qué?

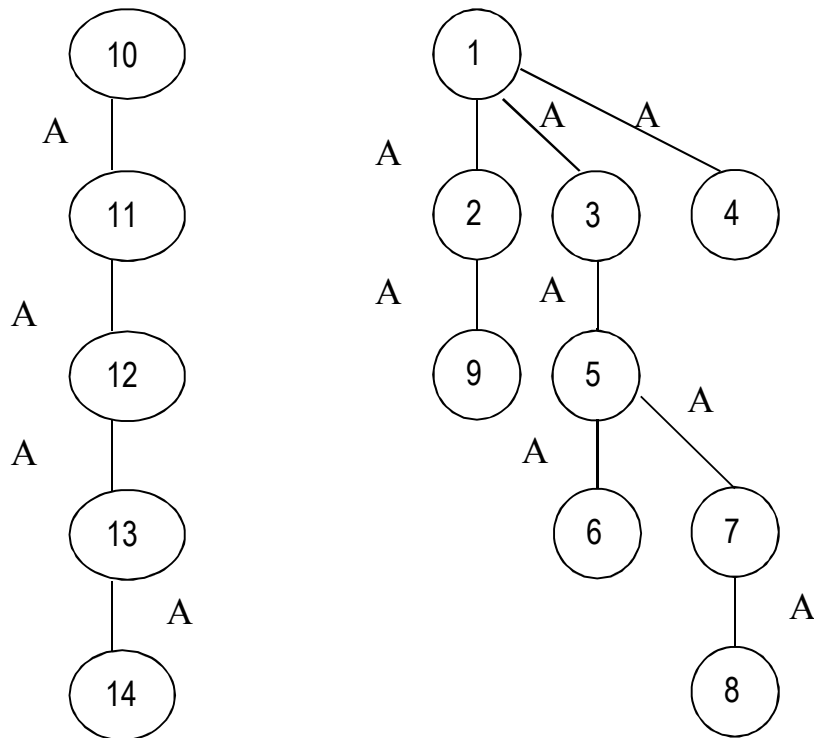
Justifica tus respuestas.

Examen PED junio 2015. Soluciones

1.

a) DFS(1)=1,2,9,3,5,6,7,8,4. Se continúa por DFS(10)=10,11,12,13,14

Árbol extendido en profundidad. Las aristas marcadas son de árbol (A), el resto son de retroceso



b) BFS(1)=1,2,3,4,5,6,7,9,8

c)

CalculaPesos: grafo \rightarrow natural

Var G: grafo; x,y: vértice; p: natural;

CalculaPesos (crear_grafo())=0

CalculaPesos (InsertarArista(G,x,y,p))=

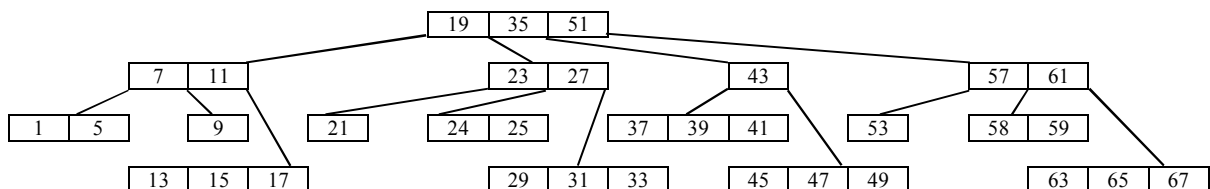
si ((x MOD 2 == 0) && (y MOD 2 == 0))

entonces p + CalculaPesos (G)

sino CalculaPesos (G)

2.

a)



b) Por ejemplo, el ejercicio 1 de los apuntes

3.

a) $\Omega(n)=1+1*n=n$ (en el caso de que entrara en el if)

$O(n)=1+2*n=n$ (en el caso de que entrara en el else)

La diferencia entre ambos casos es de sólo un paso por lo que podemos considerar que el caso mejor y peor coinciden.

- b) Si se pasara como parámetro una cadena muy larga y la suma de los códigos ASCII de sus letras superara el tamaño máximo de almacenamiento del tipo int se produciría un error.
- c) Dado que es una tabla de dispersión cerrada de tamaño B=7 como máximo se podrían insertar 7 elementos, por lo que el último elemento CASO, no se podría insertar hasta que no se eliminase otro elemento de la tabla.

2ª función hash $k(x) = (x \text{ MOD } (B-1)) + 1$
 $h_i(x) = (h_{i-1}(x) + k(x)) \text{ MOD } B$

0	COSA
1	ROMA
2	AMOR
3	ESPONJA
4	RAMO
5	JAPONES
6	MORA

AMOR: $H(303)=303 \text{ MOD } 7=2$

COSA: $H(294)=294 \text{ MOD } 7=0$

MORA: $H(303)=303 \text{ MOD } 7=2$

$k(303)=(303 \text{ MOD } 6)+1=4$

$h_1(303)=(2+4) \text{ MOD } 7=6$

ESPONJA: $H(528)=528 \text{ MOD } 7=3$

RAMO: $H(303)=303 \text{ MOD } 7=2$

$k(303)=(303 \text{ MOD } 6)+1=4$

$h_1(303)=(2+4) \text{ MOD } 7=6$

$h_2(303)=(6+4) \text{ MOD } 7=3$

$h_3(303)=(3+4) \text{ MOD } 7=0$

$h_4(303)=(0+4) \text{ MOD } 7=4$

JAPONES: $H(528)=528 \text{ MOD } 7=3$

$k(528)=(528 \text{ MOD } 6)+1=1$

$h_1(528)=(3+1) \text{ MOD } 7=4$

$h_2(528)=(4+1) \text{ MOD } 7=5$

ROMA: $H(303)=303 \text{ MOD } 7=2$

$k(303)=(303 \text{ MOD } 6)+1=4$

$h_1(303)=(2+4) \text{ MOD } 7=6$

$h_2(303)=(6+4) \text{ MOD } 7=3$

$h_3(303)=(3+4) \text{ MOD } 7=0$

$h_4(303)=(0+4) \text{ MOD } 7=4$

$h_5(303)=(4+4) \text{ MOD } 7=1$

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED junio 2016

Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 2 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En la inserción de un elemento en un árbol 2-3, la altura del árbol resultado siempre crece (con respecto al árbol original) cuando la raíz del árbol original es un 3-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	1	F
En la inserción de un elemento en un árbol 2-3-4, la altura del árbol resultado siempre crece (con respecto al árbol original) cuando la raíz del árbol original es un 4-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	2	V
En el algoritmo de borrado de un elemento en un árbol 2-3-4, siempre que el nodo “q” sea 2-nodo hay que hacer reestructuraciones.	<input type="checkbox"/>	<input type="checkbox"/>	3	V
La complejidad temporal de la operación desapilar (vista en clase) utilizando vectores (con un índice que indica la cima de la pila) o utilizando listas enlazadas es la misma.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
La semántica de la operación quita_hojas que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas es la siguiente: VAR i, d: arbin; x: ítem; quita_hojas(crea_arbin()) = crea_arbin() quita_hojas(enraizar(crea_arbin(), x, crea_arbin())) = enraizar(crea_arbin(), x, crea_arbin()) quita_hojas(enraizar(i, x, d)) = enraizar(quita_hojas(i), x, quita_hojas(d))	<input type="checkbox"/>	<input type="checkbox"/>	5	F
Todo árbol mínimo es un árbol binario de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	6	F
El grado de los árboles AVL puede ser +1, 0 ó -1.	<input type="checkbox"/>	<input type="checkbox"/>	7	F
Todo árbol binario de búsqueda es un árbol 2-3.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
En un árbol 2-3-4 el máximo número elementos del nivel N es $3 \cdot 2^{N-2}$	<input type="checkbox"/>	<input type="checkbox"/>	9	V
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C: Conjunto; x: Ítem): Operación(Crear) $\Leftrightarrow 0$ Operación (Insertar(C, x)) $\Leftrightarrow 3 + \text{Operación}(C)$	<input type="checkbox"/>	<input type="checkbox"/>	10	V
En el TAD Diccionario con dispersión cerrada, con función de redispersión “hi(x)=(H(x) + k(x)*i) 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.	<input type="checkbox"/>	<input type="checkbox"/>	11	V
En un Hash cerrado con factor de carga α , se cumple que $0 \leq \alpha \leq 1$	<input type="checkbox"/>	<input type="checkbox"/>	12	V
En un montículo doble, un elemento “j” del montículo máximo es el simétrico de un único elemento “i” del montículo mínimo.	<input type="checkbox"/>	<input type="checkbox"/>	13	F
Un multigrafo es un grafo que no tiene ninguna restricción: pueden existir arcos reflexivos y múltiples ocurrencias del mismo arco.	<input type="checkbox"/>	<input type="checkbox"/>	14	V
Sea $G=(V,A)$ un grafo dirigido. Diremos que $G''=(V'',A'')$ es un árbol extendido de $G \Leftrightarrow V''=V, A'' \subset A, \forall v \in V'' \Rightarrow \text{grado}E(v) \leq 1$	<input type="checkbox"/>	<input type="checkbox"/>	15	V

Examen PED junio 2016

Normas: ♦ Tiempo para efectuar el examen: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
- Cada pregunta se escribirá en hojas diferentes.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Cada pregunta vale 2 puntos (sobre 10).**
- Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.

1. Dado el **grafo no dirigido** representado por la lista de adyacencia que se muestra a continuación:

1 → 2 → 4
2 → 9
3 → 1 → 7
5 → 6 → 3 → 1
6 → 1
7 → 1 → 5
8 → 7
9 → 1
10 → 11
11 → 13
12 → 14 → 13
13 → 10
14 → 10 → 11

a) Obtener DFS(11), el árbol extendido en profundidad partiendo del vértice 11 y la clasificación de las aristas.

2. a) Definir la sintaxis y la semántica de la función *Camino* que recibe como parámetros una lista y un árbol binario y devuelve un booleano. Esta función devolverá TRUE si la lista de elementos pasada como parámetro forma un camino válido en el árbol binario pasado como parámetro. En otro caso devolverá FALSE. Nota: Los elementos de la lista y del árbol binario son naturales. Se podrán utilizar todas las operaciones definidas en clase para los TADs mencionados.

b) Insertar en una tabla de dispersión cerrada de tamaño $B=7$, con función de dispersión $H(x) = x \text{ MOD } B$ y con estrategia de redispersión segunda función hash, los siguientes elementos: 2341, 4123, 83, 911, 5211, 3411, 52.

Indica el número total de intentos necesarios para almacenar todos los elementos.

Nota: La información utilizada para almacenar cada elemento en la tabla HASH será la suma de las cifras de cada elemento. Por ejemplo, para almacenar el elemento 2341 tendríamos $2+3+4+1=10$, por tanto, $H(2341)=10 \text{ MOD } 7=3$. El elemento 2341 iría en la posición 3 de la tabla HASH.

3. Dada la siguiente función que ordena un vector:

```
ORD (a:vector[natural]; n: natural)
var i,j: entero; x:natural fvar
comienzo
    para i:=2 hasta n hacer
        x:=a[i]; j:=i-1
        mientras (j>0) AND (a[j]>x)
            a[j+1]:=a[j]
            j:=j-1
        fmientras
            a[j+1]:=x
    fpara
fin
```

b) Obtener BFS(11), el árbol extendido en anchura partiendo del vértice 11 y la clasificación de las aristas.

Nota: La lista de adyacencia de cada vértice se recorre de mayor a menor vértice para todos los casos del ejercicio. Las listas están desordenadas.

c) Utilizando exclusivamente las operaciones constructoras generadoras del tipo grafo, definid la semántica de la operación examen que se aplica sobre un grafo dirigido ponderado cuyos arcos están etiquetados con números naturales (es decir, el peso de los arcos son números naturales) y devuelve el número de arcos cuyo peso es igual a uno especificado. La sintaxis de la operación ‘examen’ es la siguiente:

examen: grafo, natural_peso → natural

a) Indicar razonadamente la complejidad temporal en su caso **mejor y peor**.

b) Aplicar **detalladamente** dicho algoritmo para ordenar el siguiente vector: [40, 3, 5, 9, 12, 30, 2, 35, 1].

c) Conseguir **la misma** ordenación del vector anterior mediante el algoritmo Heapsort visto en clase. NOTA: es obligatorio explicar cómo se realizan los intercambios utilizando únicamente el vector.

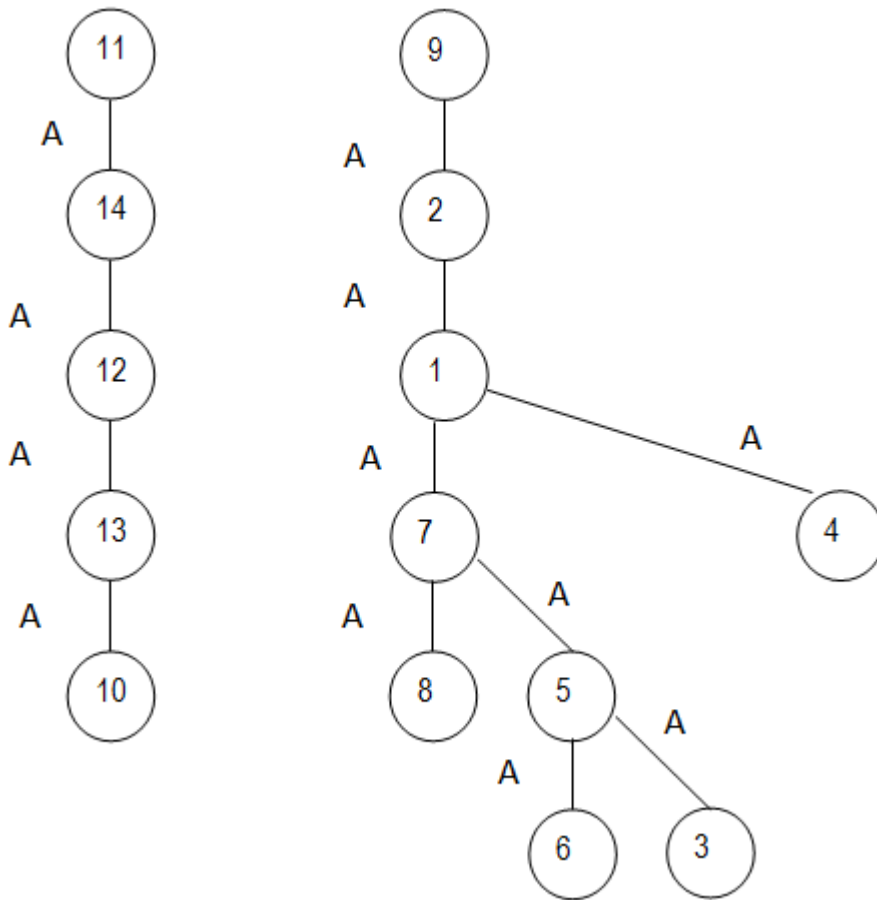
d) Calcular razonadamente la complejidad del Heapsort en su caso **mejor y peor**.

Examen PED junio 2016. Soluciones

1.

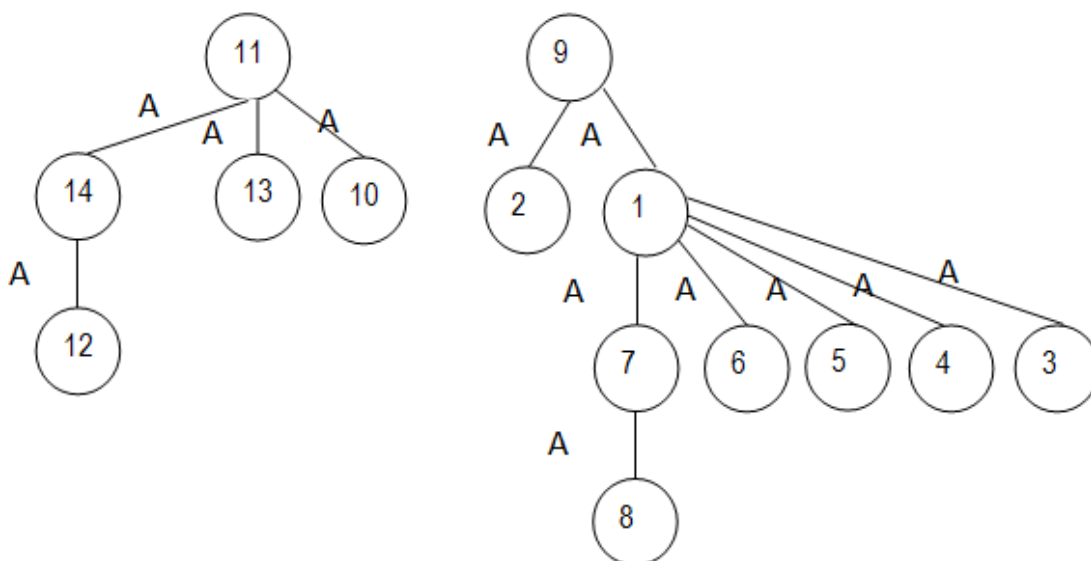
a) DFS(11)=11,14,12,13,10. Se continúa por DFS(9)=9,2,1,7,8,5,6,3,4

Árbol extendido en profundidad. Las aristas marcadas son de árbol (A), el resto son de retroceso



b) BFS(11)=11,14,13,10,12. Se continúa por BFS(9)=9,2,1,7,6,5,4,3,8

Árbol extendido en anchura. Las aristas marcadas son de árbol (A), el resto son de retroceso



c) Var G: grafo; x,y: vértice; p,q: natural;
 examen(crear_grafo(),q)=0
 examen(InsertarArista(G,x,y,p),q)=
 si (q ==p) entonces 1 + examen(G,q)
 si no examen(G,q)

2.

a)

La sintaxis de la función sería: Camino(Lista, AB) → bool
 VAR x,y: Natural; L:Lista; i,d: AB

```
Camino(crear(), crear()) = TRUE
Camino(crear(), enraizar(i, x, d) = TRUE
Camino(inscabeza(L,x), crear()) = FALSE
Camino(inscabeza(L,x), enraizar(i, y, d)) =
  Si (x!=y) entonces Camino(inscabeza(L, x), i) OR Camino(inscabeza(L, x), d) Fsi
    Si (x==y) entonces
      Si (es_vacia(L) entonces TRUE
      Sino
        Si (obtener(L, primera(L)) == raiz(i)) entonces Camino(L, i)
        Sino
          Si (obtener(L, primera(L)) == raiz(d) entonces Camino(L, d)
          Sino FALSE
        Fsi
      Fsi
    Fsi
  Fsi
```

b)

$$K(x) = (x \text{ MOD } (B-1)) + 1$$

$$h_i(x) = (h_{i-1}(x) + k(x)) \text{ MOD } B$$

$$H(2341) = 10 \text{ MOD } 7 = 3$$

$$H(4123) = 10 \text{ MOD } 7 = 3$$

$$k(4123) = (10 \text{ MOD } 6) + 1 = 5$$

$$h_1(4123) = (3+5) \text{ MOD } 7 = 1$$

$$H(83) = 11 \text{ MOD } 7 = 4$$

$$H(911) = 11 \text{ MOD } 7 = 4$$

$$k(911) = (11 \text{ MOD } 6) + 1 = 6$$

$$h_1(911) = (4 + 6) \text{ MOD } 7 = 3$$

$$h_2(911) = (3 + 6) \text{ MOD } 7 = 2$$

$$H(5211) = 9 \text{ MOD } 7 = 2$$

$$k(5211) = (9 \text{ MOD } 6) + 1 = 4$$

$$h_1(5211) = (2 + 4) \text{ MOD } 7 = 6$$

$$H(3411) = 9 \text{ MOD } 7 = 2$$

$$k(3411) = (9 \text{ MOD } 6) + 1 = 4$$

$$h_1(3411) = (2 + 4) \text{ MOD } 7 = 6$$

$$h_2(3411) = (6 + 4) \text{ MOD } 7 = 3$$

$$h_3(3411) = (3 + 4) \text{ MOD } 7 = 0$$

$$H(52) = 7 \text{ MOD } 7 = 0$$

$$k(52) = (7 \text{ MOD } 6) + 1 = 2$$

$$h_1(52) = (0 + 2) \text{ MOD } 7 = 2$$

$$h_2(52) = (2 + 2) \text{ MOD } 7 = 4$$

$$h_3(52) = (4 + 2) \text{ MOD } 7 = 6$$

$$h_4(52) = (6 + 2) \text{ MOD } 7 = 1$$

$$h_5(52) = (1 + 2) \text{ MOD } 7 = 3$$

0	3411
1	4123
2	911
3	2341
4	83
5	52
6	5211

Número de intentos totales = 20

$$h_6(52) = (3 + 2) \text{ MOD } 7 = 5$$

3.

a) Caso mejor $\Omega(n)$: en el mientras no entra nunca. El último elemento de destino es siempre el menor y como destino ya está ordenado \rightarrow vector ordenado ascendentemente

$$\sum_{i=2..n} 1 = n-1 \in \Omega(n)$$

Caso peor $O(n^2)$: todos son mayores en destino. Hay que desplazarlos todos \rightarrow vector ordenado inverso.

$$\sum_{i=2..n} (1 + \sum_{j=1..i-1} 1) = \sum_{i=2..n} (1+i-1) \in O(n^2)$$

c) Puesto que se realiza una ordenación de menor a mayor, habrá de utilizarse un montículo máximo.

d) $\Omega(n \log_2 n) = O(n \log_2 n)$ Ya que aunque esté ordenado previamente, en la creación/borrado del montículo ha de hacer n operaciones, cada una con un coste de $\log_2 n$ (la altura del montículo).

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED enero 2006

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **35 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En la especificación de un TAD, las operaciones auxiliares son visibles para los usuarios.	<input type="checkbox"/>	<input type="checkbox"/>	1.	F
Una operación del TAD X que tenga la sintaxis Crear() →X es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2.	V
En C++, los miembros <i>protected</i> son privados para el exterior, pero permiten el acceso a las clases derivadas.	<input type="checkbox"/>	<input type="checkbox"/>	3.	V
Dadas las clases <i>TDir</i> y <i>TVectorDir</i> : <pre>class TDir { public: private: int e1; char c1; }; class TVectorDir { public: private: TDir *vector; int dim; }; TVectorDir::~TVectorDir () { if (v!=NULL) delete v; dim=0; v=NULL; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	4.	F
¿Es correcta la implementación del destructor de <i>TVectorDir</i> ?	<input type="checkbox"/>	<input type="checkbox"/>	5.	F
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input type="checkbox"/>	6.	V
Dado un único recorrido de un árbol binario lleno es posible reconstruir dicho árbol	<input type="checkbox"/>	<input type="checkbox"/>	7.	F
Sea el tipo <i>arbin</i> definido en clase. La semántica de la operación <i>nodos</i> es la siguiente: <pre>Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)</pre>	<input type="checkbox"/>	<input type="checkbox"/>	8.	F
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	9.	V
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	10.	V
Un árbol completo siempre está balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	11.	F
En un árbol AVL siempre que se inserte una etiqueta hay que realizar una rotación	<input type="checkbox"/>	<input type="checkbox"/>	12.	V
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	<input type="checkbox"/>	<input type="checkbox"/>	13.	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	14.	V
La semántica de la operación <i>anterior</i> vista en clase es la siguiente: <pre>VAR L1: lista; x: item; p: posicion; anterior(L1, primera(L1)) = error_posicion(); si p != ultima(L1) entonces anterior(L1, siguiente(L1, p)) = p anterior(inscabeza(L1, x), primera(L1)) = primera(inscabeza(L1, x))</pre>	<input type="checkbox"/>	<input type="checkbox"/>	15.	F
Sea el tipo <i>vector</i> definido en clase. La semántica de la operación <i>recu</i> es la siguiente: <pre>Var v:vector; i,j:int; x:item; recu(crear_vector(),i)=error_item() si i > j entonces recu(asig(v,i,x),j)=recu(v,j) sino recu(asig(v,i,x),j)=TRUE</pre>	<input type="checkbox"/>	<input type="checkbox"/>	16.	V
<i>crear_pila()</i> , <i>apilar(pila,item)</i> y <i>desapilar(pila)</i> son operaciones constructoras del tipo <i>pila</i> .	<input type="checkbox"/>	<input type="checkbox"/>	17.	F
Todo árbol binario de búsqueda es un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	18.	F
La semántica de la operación <i>Base</i> que actúa sobre una <i>pila</i> y devuelve el primer elemento apilado es la siguiente: <pre>Base(crear_pila ()) = crear_pila () Base(apilar(crear_pila (), x)) = x Base(apilar(p, x)) = Base(p)</pre>	<input type="checkbox"/>	<input type="checkbox"/>		

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED febrero 2007

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **35 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En C++ y cuando se emplea composición (<i>layering</i>), los métodos de la clase derivada pueden acceder a la parte pública de la clase base.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
En C++, el puntero <i>this</i> no se puede emplear para modificar el objeto al que apunta.	<input type="checkbox"/>	<input type="checkbox"/>	2	F
En C++, los constructores se pueden invocar explícitamente cuando el programador lo desee (por ejemplo: <i>TLista a; a.TLista();</i>).	<input type="checkbox"/>	<input type="checkbox"/>	3	F
En C++, la siguiente declaración es incorrecta: $int\& a = 1;$	<input type="checkbox"/>	<input type="checkbox"/>	4	V
En la escala de complejidades se cumple que $O(\log n) \subset O(\log \log n)$.	<input type="checkbox"/>	<input type="checkbox"/>	5	F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\mathcal{O}(1)$.	<input type="checkbox"/>	<input type="checkbox"/>	6	V
Para un vector de naturales, se define la operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con "0" (para calcular el resto de una división, se puede utilizar la operación MOD). La sintaxis y la semántica de la operación <i>eliminar</i> es la siguiente: <i>eliminar</i> : $vector \rightarrow vector$ <i>Var</i> <i>v</i> :vector; <i>i</i> : entero; <i>x</i> :natural; <i>eliminar</i> (<i>crear_vector</i> ()) = <i>crear_vector</i> () <i>si</i> (<i>i</i> MOD 2) == 0 <i>entonces eliminar</i> (<i>asignar</i> (<i>v</i> , <i>i</i> , <i>x</i>)) = <i>asignar</i> (<i>eliminar</i> (<i>v</i>), <i>i</i> ,0) <i>si no eliminar</i> (<i>asignar</i> (<i>v</i> , <i>i</i> , <i>x</i>)) = <i>asignar</i> (<i>eliminar</i> (<i>v</i>), <i>i</i> , <i>x</i>)	<input type="checkbox"/>	<input type="checkbox"/>	7	V
La semántica de la operación <i>ultima</i> vista en clase es la siguiente: <i>VAR</i> <i>L1</i> : lista; <i>x</i> : item; <i>si</i> <i>esvacia</i> (<i>L1</i>) <i>entonces</i> <i>ultima</i> (<i>inscabeza</i> (<i>L1</i> , <i>x</i>) = <i>primera</i> (<i>L1</i>) <i>si no ultima</i> (<i>inscabeza</i> (<i>L1</i> , <i>x</i>) = <i>ultima</i> (<i>L1</i>)	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en preorden con 62 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/>	9	F
La sintaxis y semántica de la operación <i>quita_hojas</i> , que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas, es la siguiente: <i>quita_hojas</i> (<i>arbin</i>) \rightarrow <i>arbin</i> <i>VAR</i> <i>i</i> , <i>d</i> : <i>arbin</i> ; <i>x</i> : item; <i>quita_hojas</i> (<i>crea_arbin</i> ()) = <i>crea_arbin</i> () <i>quita_hojas</i> (<i>enraizar</i> (<i>crea_arbin</i> (), <i>x</i> , <i>crea_arbin</i> ()) = <i>crea_arbin</i> () <i>quita_hojas</i> (<i>enraizar</i> (<i>i</i> , <i>x</i> , <i>d</i>)) = <i>enraizar</i> (<i>quita_hojas</i> (<i>i</i>), <i>x</i> , <i>quita_hojas</i> (<i>d</i>))	<input type="checkbox"/>	<input type="checkbox"/>	10	V
Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 2 y 1 hoja es posible reconstruir 2 árboles binarios.	<input type="checkbox"/>	<input type="checkbox"/>	11	V
Todo árbol AVL es un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	12	F
La operación (<i>DIVIDEHIJODE2</i> (<i>p</i> , <i>q</i>)) en la inserción de un elemento en un árbol 2-3-4 puede aumentar la altura del árbol original.	<input type="checkbox"/>	<input type="checkbox"/>	13	F
En el algoritmo del borrado de un elemento en un árbol 2-3-4 si <i>q</i> es 2-nodo y <i>r</i> es 3-nodo hay que hacer una ROTACIÓN.	<input type="checkbox"/>	<input type="checkbox"/>	14	V

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED febrero 2008

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **30 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
Dada la sintaxis de la función $IC(lista, item) \rightarrow lista$, que inserta un elemento a la cabeza de la lista pasada como parámetro y $crear() \rightarrow lista$, que crea una lista vacía. La siguiente secuencia: $IC(IC(IC(crear(), a), b), c)$, daría como resultado una lista con los elementos en este orden: $a \rightarrow b \rightarrow c$, donde a es el primer elemento de la lista	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: $F: natural \rightarrow BOOL$, y su semántica como: $F(cero)=TRUE$, $F(suc(cero))=FALSE$, $F(suc(suc(x)))=F(x)$. Para el número natural $x=35$, la función F devolvería $TRUE$.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	<input type="checkbox"/>	<input type="checkbox"/>	5	F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$.	<input type="checkbox"/>	<input type="checkbox"/>	6	V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución	<input type="checkbox"/>	<input type="checkbox"/>	7	V
La operación <code>BorrarItem</code> , que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semántica: BorrarItem: LISTA, ITEM \rightarrow LISTA BorrarItem(Crear, i) = Crear BorrarItem(IC(L1, j), i) = si ($i == j$) entonces BorrarItem (L1, i) sino IC (BorrarItem (L1, i), j)	<input type="checkbox"/>	<input type="checkbox"/>	8	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)	<input type="checkbox"/>	<input type="checkbox"/>	9	F
El máximo número de nodos en un nivel $i-1$ de un árbol binario es 2^{i-2} , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	10	V
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente: Var i, d: arbin; x: item; nodos(crear_arbin())=0 nodos(enraizar(i, x, d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	11	F
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	12	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	13	V

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED febrero 2008

Modalidad 1

- Normas:**
- La entrega del test no corre convocatoria.
 - * Tiempo para efectuar el test: **30 minutos**.
 - * Una pregunta mal contestada elimina una correcta.
 - * Las soluciones al examen se dejarán en el campus virtual.
 - * **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - * En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	2	V
Dada la sintaxis de la función $IC(lista, item) \rightarrow lista$, que inserta un elemento a la cabeza de la lista pasada como parámetro y $crear() \rightarrow lista$, que crea una lista vacía. La siguiente secuencia: $IC(IC(IC(crear(), a), b), c)$, daría como resultado una lista con los elementos en este orden: $a \rightarrow b \rightarrow c$, donde a es el primer elemento de la lista	<input type="checkbox"/>	<input type="checkbox"/>	3	F
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: $F: natural \rightarrow BOOL$, y su semántica como: $F(cero)=TRUE$, $F(suc(cero))=FALSE$, $F(suc(suc(x)))=F(x)$. Para el número natural $x=35$, la función F devolvería $TRUE$.	<input type="checkbox"/>	<input type="checkbox"/>	4	F
El máximo número de nodos en un nivel $i-1$ de un árbol binario es 2^{i-2} , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	5	V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	<input type="checkbox"/>	<input type="checkbox"/>	6	F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	7	V
La operación <code>BorrarItem</code> , que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semántica: BorrarItem: LISTA, ITEM \rightarrow LISTA BorrarItem(Crear, i) = Crear BorrarItem(IC(L1, j), i) = si ($i == j$) entonces BorrarItem (L1, i) sino IC (BorrarItem (L1, i), j)	<input type="checkbox"/>	<input type="checkbox"/>	8	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)	<input type="checkbox"/>	<input type="checkbox"/>	9	F
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución	<input type="checkbox"/>	<input type="checkbox"/>	10	V
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	11	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	12	V
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente: Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	13	F

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED febrero 2008

Modalidad 2

- Normas:**
- La entrega del test no corre convocatoria.
 - * Tiempo para efectuar el test: **30 minutos**.
 - * Una pregunta mal contestada elimina una correcta.
 - * Las soluciones al examen se dejarán en el campus virtual.
 - * **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - * En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: $F: \text{natural} \rightarrow \text{BOOL}$, y su semántica como: $F(\text{cero}) = \text{TRUE}$, $F(\text{suc}(\text{cero})) = \text{FALSE}$, $F(\text{suc}(\text{suc}(x))) = F(x)$. Para el número natural $x=35$, la función F devolvería TRUE.	<input type="checkbox"/>	<input type="checkbox"/>	1	F
La operación <code>BorrarItem</code> , que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semántica: BorrarItem: LISTA, ITEM \rightarrow LISTA BorrarItem(Crear, i) = Crear BorrarItem(IC(L1,j), i) = si (i == j) entonces BorrarItem (L1, i) sino IC (BorrarItem (L1, i), j)	<input type="checkbox"/>	<input type="checkbox"/>	2	V
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	3	V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución	<input type="checkbox"/>	<input type="checkbox"/>	4	V
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	5	V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	<input type="checkbox"/>	<input type="checkbox"/>	6	F
El máximo número de nodos en un nivel i-1 de un árbol binario es 2^{i-2} , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	7	V
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente: Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	9	V
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	10	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)	<input type="checkbox"/>	<input type="checkbox"/>	11	F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$.	<input type="checkbox"/>	<input type="checkbox"/>	12	V
Dada la sintaxis de la función $IC(\text{lista}, \text{item}) \rightarrow \text{lista}$, que inserta un elemento a la cabeza de la lista pasada como parámetro y $\text{crear}() \rightarrow \text{lista}$, que crea una lista vacía. La siguiente secuencia: $IC(IC(IC(\text{crear}(),a),b),c)$, daría como resultado una lista con los elementos en este orden: $a \rightarrow b \rightarrow c$, donde a es el primer elemento de la lista	<input type="checkbox"/>	<input type="checkbox"/>	13	F

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen TAD/PED julio 2004

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - **El test vale un 40% de la nota de teoría.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F						
La operación crear_pila() es constructora modificadora.	<input type="checkbox"/>	<input type="checkbox"/>	F 1.					
En C++, una forma correcta de copiar una cadena es la siguiente: <i>char a[50] = “Tipos Abstractos de Datos”;</i> <i>char *b;</i> <i>b = new char[strlen(a)];</i> <i>strcpy(b, a);</i>	<input type="checkbox"/>	<input type="checkbox"/>	F 2.					
El caso peor de la búsqueda es más eficiente en una lista ordenada que en una lista cuyos elementos no están ordenados.	<input type="checkbox"/>	<input type="checkbox"/>	F 3.					
En un árbol binario cada elemento puede tener como máximo dos predecesores.	<input type="checkbox"/>	<input type="checkbox"/>	F 4.					
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.	<input type="checkbox"/>	<input type="checkbox"/>	F 5.					
Dado un recorrido en preorden (RID) de un árbol AVL es posible reconstruir un único árbol AVL.	<input type="checkbox"/>	<input type="checkbox"/>	V 6.					
El número máximo de elementos que se puede almacenar en un árbol 2-3 de altura h es 3 ^h -1	<input type="checkbox"/>	<input type="checkbox"/>	V 7.					
Al borrar un elemento en un árbol 2-3-4 se puede realizar una operación de DIVIDERAIZ.	<input type="checkbox"/>	<input type="checkbox"/>	F 8.					
En un árbol B m-camino de búsqueda con m=16: en cualquier nodo excepto la raíz hay 8 ítems como mínimo.	<input type="checkbox"/>	<input type="checkbox"/>	F 9.					
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: Ítem): Insertar(Insertar(C, x), y) ⇔ Insertar(Insertar(C,y), x)	<input type="checkbox"/>	<input type="checkbox"/>	V 10.					
En el TAD Diccionario con dispersión cerrada, con función de redispersión “h _i (x)=(H(x) + k(x)*i) 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.	<input type="checkbox"/>	<input type="checkbox"/>	V 11.					
El siguiente vector representa un montículo máximo: <table border="1"><tr><td>10</td><td>5</td><td>3</td><td>1</td><td>2</td></tr></table>	10	5	3	1	2	<input type="checkbox"/>	<input type="checkbox"/>	V 12.
10	5	3	1	2				
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.	<input type="checkbox"/>	<input type="checkbox"/>	F 13.					
La complejidad temporal de la búsqueda en un trie es lineal respecto al número de palabras almacenadas	<input type="checkbox"/>	<input type="checkbox"/>	F 14.					
Un bosque extendido en profundidad de un grafo no dirigido es un grafo acíclico.	<input type="checkbox"/>	<input type="checkbox"/>	V 15.					

Examen TAD/PED julio 2004

Normas: ♦ Tiempo para efectuar el ejercicio: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: *Apellidos, Nombre*. Cada pregunta se escribirá en folios diferentes.
- Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
- **Publicación de notas de exámenes y prácticas:** 6 de julio. **ÚNICA fecha de revisión de exámenes y prácticas:** 9 de julio. El lugar y la hora se publicará en el campus virtual.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Sean 2 leftist mínimos representados como árboles binarios (se asume que no están vacíos y que no hay elementos repetidos) y cuyas etiquetas son números naturales. Utilizando exclusivamente las operaciones constructoras generadoras de los árboles binarios definir la sintaxis y la semántica de la primera parte de la operación *combinar* en la que se crea un árbol resultado de la combinación de ambos, sin comprobar la condición de leftist y actualización de caminos mínimos.
2. El siguiente fragmento de código define una clase en C++ que representa una lista doblemente enlazada de punteros a objetos TPoros. Se pide el código del destructor, el constructor de copia y la sobrecarga del operador de asignación (en éstos hay que duplicar la memoria de modo que si se hace una asignación y se destruye una lista la otra no se destruya).

```
class TNode {
    TPoros *dato;
    TNode *sig, *ant;
};
class TLista {
public:
    TLista();
    TLista(TPoros *);
    ...
private:
    TNode *primero, *ultimo;
};
```

3. Sea un árbol B de orden $m=6$ inicialmente vacío:
 - a) Insertar los siguientes elementos 10, 5, 7, 3, 12, 6, 20, 30, 35, 40, 45, 41.
 - b) Del árbol B resultante, borrar los elementos 12, 45.
 - c) Del árbol B resultante, insertar los elementos 2, 4, 8, 9, 21, 22, 23, 25, 26, 27, 28, 29, 33, 34, y después borrar 8 y 9.
 - d) El árbol B resultante cumple también las propiedades para ser un árbol 234. Enumera cuáles son estas propiedades e inserta el ítem 31 utilizando el algoritmo de inserción del árbol 234.

Reglas a tener en cuenta:

1. En la inserción cuando hagamos la división de nodos dejaremos la menor cantidad de elementos posibles en el nodo de la izquierda.
2. Al realizar un borrado de un ítem interior (no hoja), lo sustituiremos por el mayor de la izquierda.
3. Al realizar un borrado de un ítem, elegiremos como nodo adyacente el nodo de la izquierda. En el caso de que el nodo objeto de la transformación sea el nodo más a la izquierda, consideraremos el nodo derecha de éste como nodo adyacente.

4. a) Sobre un *trie* con nodos terminales como el del ejemplo, calcular las cotas superiores de complejidad temporal en su caso peor y mejor, para las siguientes operaciones:

Insertar(palabra),

Insertar(conjuntoDePalabras) (por ejemplo conjuntoDePalabras={1,123,AB})

ListarPalabrasConPrefijo(prefijo) (del trie ejemplo, con *prefijo*=234, esta función

devolvería {234,2349,23495})

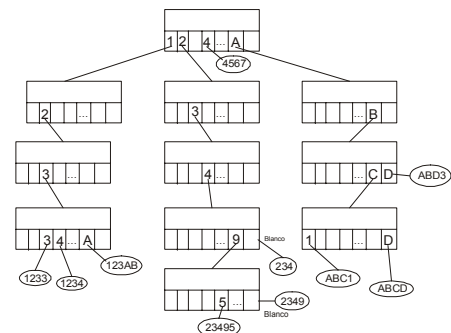
NumeroDeElementos() (del trie ejemplo esta función devolvería el valor 10).

Para ello utilizar los siguientes parámetros: n (longitud de "palabra"), m (longitud de "prefijo"), k (número de letras del alfabeto o tamaño del vector), L (longitud de la palabra más larga posible), T (tamaño de "conjuntoDePalabras"), p (número de palabras del diccionario).

- b) Sea un grafo no dirigido de n vértices numerados de 1 a n , representado con una lista de adyacencia, en la que sólo se almacenan las aristas (i, j) tal que $i < j$, con i y j de tipo "vértice". Calcular las cotas superiores de complejidad temporal en su caso peor y

mejor, para las siguientes operaciones: *Insertar(i, j)*, *Adyacencia(i)*, *NumeroDeAristas()*.

NOTA: Hay que explicar cada complejidad con un máximo de 3 líneas. Si la explicación no es correcta, no se valorará positivamente.



Examen TAD/PED diciembre 2003. Soluciones

1)

combinar: arbin, arbin \rightarrow arbin

Var i,l,d,r:arbin; x,y:natural;

combinar(crear_arbin(),enraizar(i,x,d)) = enraizar(i,x,d)

si $x < y$

entonces combinar(enraizar(i,x,d), enraizar(l,y,r)) = enraizar(i,x,combinar(d,enraizar(l,y,r)))

sino combinar(enraizar(i,x,d), enraizar(l,y,r)) = enraizar(l,y,combinar(r,enraizar(i,x,d)))

2)

a) Destructor

```
TLista::~~TLista()
{
    TNode *aux;

    while(primero != NULL)
    {
        aux = primero;
        primero = primero->sig;
        delete aux->dato;
        delete aux;
    }
    primero = ultimo = NULL;
}
```

b) Constructor de copia

```
TLista::TLista(TLista &l)
{
    TNode *a;
    TNode *aux = l.primero;

    if(aux != NULL)
    {
        a = new TNode;
        a->dato = new TPoro(*(aux->dato));
        a->sig = NULL;
        a->ant = NULL;
        primero = ultimo = a;

        aux = aux->sig;
        while(aux != NULL)
        {
            a = new TNode;
            a->dato = new TPoro(*(aux->dato));
            a->sig = NULL;
            a->ant = ultimo;
            ultimo = a;
            aux = aux->sig;
        }
    }
    else
        primero = ultimo = NULL;
}
```

Otra alternativa:

```
TLista::TLista(TLista &l)
{
    primero = ultimo = NULL;
    *this = l;
}
```

```

}
```

c) Sobrecarga del operador asignación

```

TLista&
TLista::operator=(TLista &l)
{
    TNode *a;
    TNode *aux = l.primer;

    if(this == &l)
        return *this;

    this->~TLista();
    if(aux != NULL)
    {
        a = new TNode;
        a->dato = new TPoro(*(aux->dato));
        a->sig = NULL;
        a->ant = NULL;
        primero = ultimo = a;

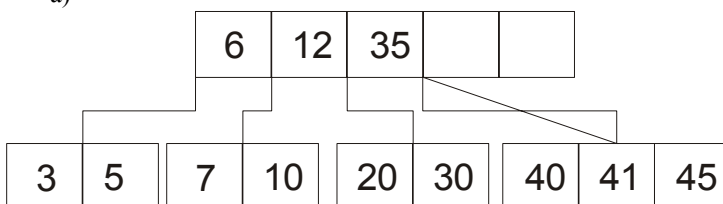
        aux = aux->sig;
        while(aux != NULL)
        {
            a = new TNode;
            a->dato = new TPoro(*(aux->dato));
            a->sig = NULL;
            a->ant = ultimo;
            ultimo = a;
            aux = aux->sig;
        }
    }
    else
        primero = ultimo = NULL;

    return *this;
}

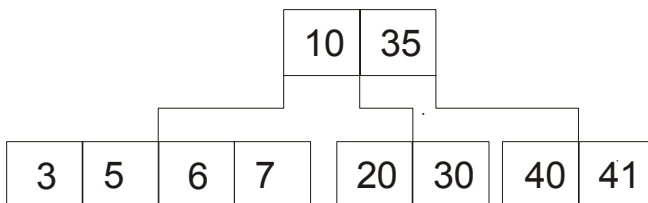
```

3)

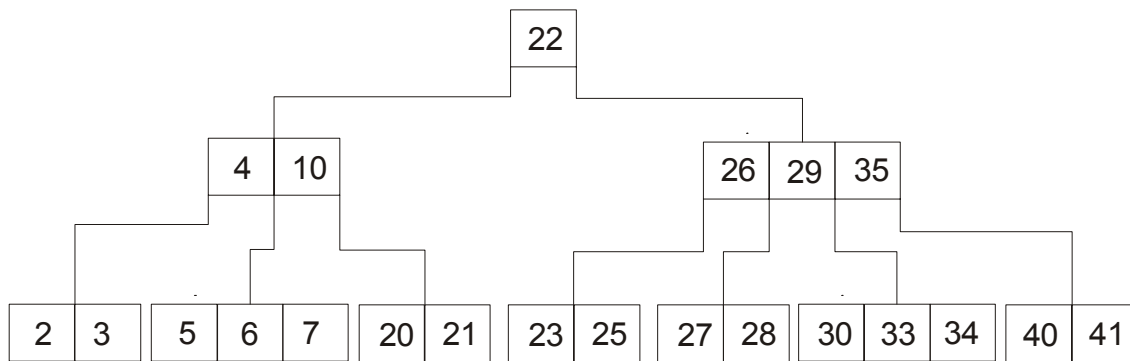
a)



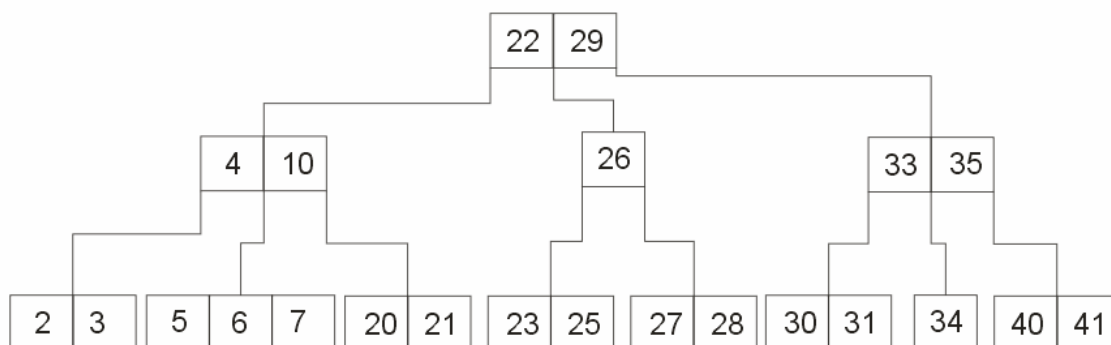
b)



c)



d)



4) a)

	Peor Caso	Mejor Caso
Insertar(palabra)	$O(n)$ hay que crear un nodo no terminal por cada letra de "palabra"	$\Omega(1)$ cuando sólo hay que añadir un nodo terminal, por ejemplo para <i>insertar(BCD)</i>
Insertar(conjunto DePalabras)	$O(L \cdot T)$ misma explicación que el caso anterior para las T palabras de <i>conjuntoDePalabras</i>	$\Omega(T)$ misma explicación que el caso anterior para las T palabras de <i>conjuntoDePalabras</i>
ListarPalabrasConPrefijo(prefijo)	$O(m+k \cdot (L-m))$ cada letra del prefijo aparecerá como nodo no terminal, y habrá que recorrer cada posición del vector del nodo terminal de su última letra, y como peor caso cada palabra tendrá longitud L , con cada letra representada con nodos no terminales.	$\Omega(1)$ caso que sólo hay una palabra con ese prefijo en todo el diccionario, y ninguna otra palabra que comparta ninguna otra letra del prefijo. ó $\Omega(m+k)$ caso que haya que recorrer cada letra del prefijo como nodo no terminal y recorrer el vector de ese nodo, en el que cada palabra que tenga ese prefijo esté representada por un único nodo terminal
NumeroDeElementos()	$O(L \cdot p)$ cada una de las " p " palabras del diccionario, todas sus letras estarán representadas con nodos no terminales, con una longitud máxima de palabra " L "	$\Omega(p+k)$ cada palabra estará representada por un único nodo no terminal

b)

	Peor Caso	Mejor Caso
Insertar(i, j), suponiendo que $i < j$	$O(n)$ hay que recorrer la lista del vértice " i ", y el caso peor es que esa lista tenga " $n-1$ " vértices	$\Omega(1)$ cuando la lista del vértice " i " está vacía
Adyacencia (i)	$O(\sum_{j=1}^i (n-j)) = O(n^2)$, cada lista " j " del vector tendrá " $n-j$ " vértices como máximo (caso peor), y habrá que recorrer todas las listas de la posición " 1 " del vector hasta " i "	$\Omega(i)$ cada lista del vector desde " 1 " hasta " i " estará vacía

NumeroDeAristas()	$O(\frac{n * (n - 1)}{2}) = O(n^2)$ máximo número de aristas en un grafo no dirigido	$\Omega(n)$ grafo vacío, en el que habrá que recorrer secuencialmente cada casilla del vector
--------------------------	---	--

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen TAD/PED Junio 2005

Modalidad 0

Normas:

- La entrega del test no corre convocatoria.
- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
- **El test vale un 40% de la nota de teoría.**
- En la **HOJA DE CONTESTACIONES** el verdadero se corresponderá con la **A** y el falso con la **B**.

	V	F	
Siempre que se realiza una rotación DD en el borrado de un elemento en un árbol AVL decrece la altura del subárbol sobre el que se realiza la rotación.	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Un árbol 2-3 es un árbol m -camino de búsqueda con $m=3$.	<input type="checkbox"/>	<input type="checkbox"/>	2 V
La complejidad temporal (en su peor caso) de buscar un elemento en un vector ordenado utilizando un algoritmo de búsqueda binaria es $O(\log_n 2)$ [siendo n el número de elementos].	<input type="checkbox"/>	<input type="checkbox"/>	3 F
Un árbol rojo-negro, en el que no hay ningún enlace rojo, es un árbol binario lleno	<input type="checkbox"/>	<input type="checkbox"/>	4 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.	<input type="checkbox"/>	<input type="checkbox"/>	5 V
En el algoritmo HeapSort, después del primer paso de insertar todos los elementos en un Heap representado como un vector, los elementos del mismo quedan ordenados.	<input type="checkbox"/>	<input type="checkbox"/>	6 F
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.	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Sea $G=(V,A)$ un grafo dirigido. Diremos que $G''=(V'',A'')$ es un árbol extendido de $G \Leftrightarrow V''=V, A'' \subset A, \forall v \in V'' \Rightarrow \text{grado}_E(v) \leq 1$.	<input type="checkbox"/>	<input type="checkbox"/>	8 V
La operación <i>BorrarItem</i> , que borra la primera ocurrencia del item i que se encuentre en la lista, tiene la siguiente sintaxis y semántica: BorrarItem: LISTA, ITEM \rightarrow LISTA BorrarItem(Crear, i) = Crear BorrarItem(IC(L1, j), i) = si ($i == j$) entonces BorrarItem (L1, i) sino IC (BorrarItem (L1, i), j)	<input type="checkbox"/>	<input type="checkbox"/>	9 F
Se puede reconstruir un único árbol binario de búsqueda teniendo sus recorridos en preorden y postorden.	<input type="checkbox"/>	<input type="checkbox"/>	10 V
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 predecesor.	<input type="checkbox"/>	<input type="checkbox"/>	11 F
Al realizar un recorrido en inorden de un montículo obtenemos una sucesión de claves ordenadas.	<input type="checkbox"/>	<input type="checkbox"/>	12 F
La función de redispersión en una tabla hash abierta tiene que cumplir como condición para que se recorran todas las posiciones del vector que el valor de B sea primo	<input type="checkbox"/>	<input type="checkbox"/>	13 F
Un grafo que tiene componentes fuertemente conexas es un grafo necesariamente libre de ciclos	<input type="checkbox"/>	<input type="checkbox"/>	14 F

Examen TAD/PED junio 2005

Normas: ♦ Tiempo para efectuar el ejercicio: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: *Apellidos, Nombre*. Cada pregunta se escribirá en folios diferentes.
- Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
- **Publicación de notas de exámenes:** 4-Julio. **Fecha de revisión de exámenes y de prácticas:** 8 de Julio. Hora de 15:00 a 17:00

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

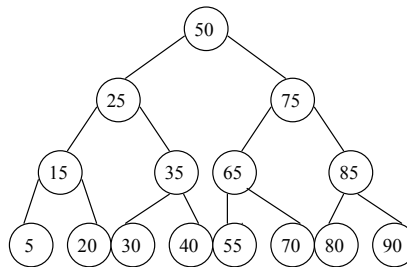
1. Utilizando exclusivamente las operaciones constructoras generadoras del vector, definid la sintaxis y la semántica de la operación ‘inversa’ que crea el vector inverso de un vector dado.

Nota: Se asume que el vector exclusivamente tiene 100 posiciones y el rango de las mismas es de 1...100, es de números naturales, y para éstos están definidas las operaciones de ‘+’, ‘-’, ‘*’ y ‘/’.

2. En el siguiente árbol 2-3-4 borrar los elementos: 30, 15 y 20. (Primero se borra el 30, sobre el resultado obtenido se borra el 15 y, finalmente, sobre el resultado obtenido se borra el 20).

Criterios: Si hay que borrar un nodo con 2 hijos hay que sustituir por el menor de la derecha. Se tomará como *r* el hermano de la derecha.

Nota: Realizar el borrado de los elementos paso a paso indicando las transformaciones realizadas.



3. En un árbol izquierdista mínimo (leftist mínimo) inicialmente vacío, insertar los siguientes items: 7, 15, 11, 1, 2, 9, 3, 13, 6, 12, 4, 22, 10, 14, 5, 30, 21, 20, 23.

Del leftist minino anterior, realizar dos operaciones de borrado.

4. Calcula la complejidad temporal en el caso peor de las siguientes operaciones utilizando las estructuras que se indican.

Utiliza la Notación asintótica O (big-omicron).

El número de elementos le llamamos *n*.

No se permiten elementos repetidos.

Razona brevemente tu respuesta.

- Inserción y borrado en una lista enlazada no ordenada.
- Inserción y borrado en un vector ordenado circular.
- Inserción de una palabra en un trie con nodos terminales.
- Obtener mínimo de un heap mínimo.
- Insertar en un árbol binario de búsqueda enlazado.
- Borrar en un árbol leftist.

Examen TAD/PED junio 2005. Soluciones

1)

Sintaxis:

inversa: vector \rightarrow vector

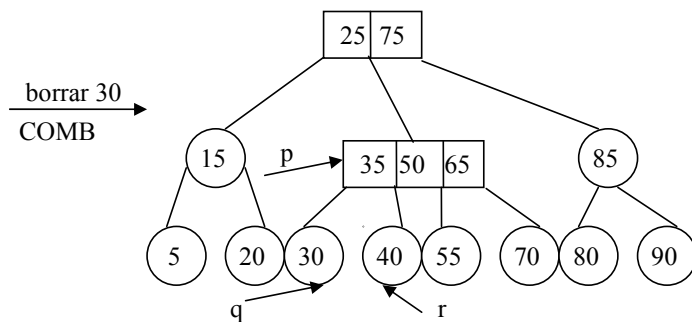
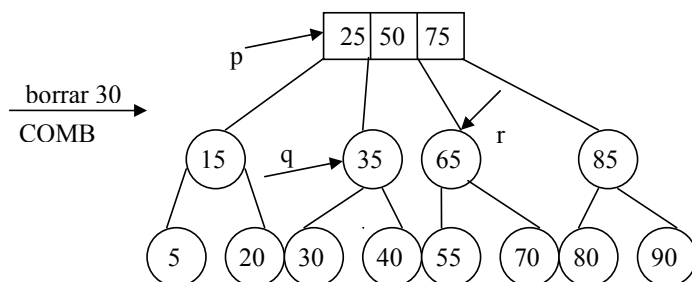
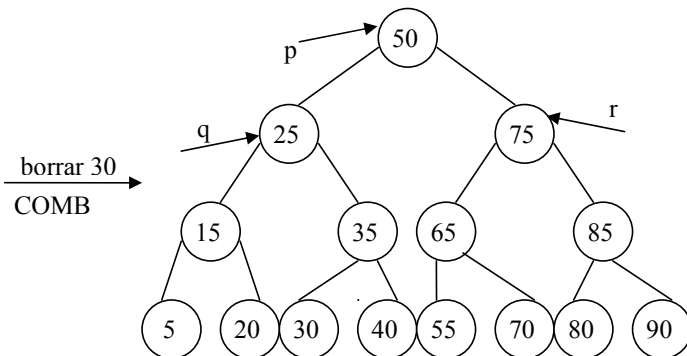
Semántica:

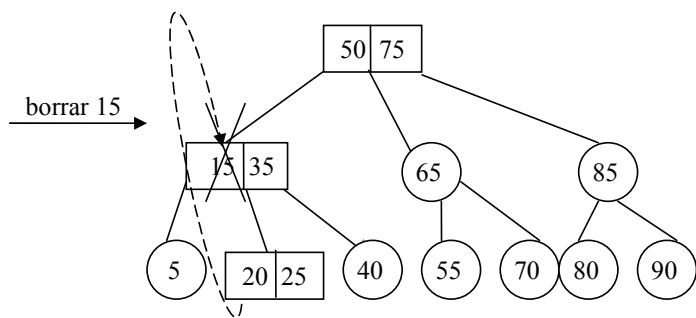
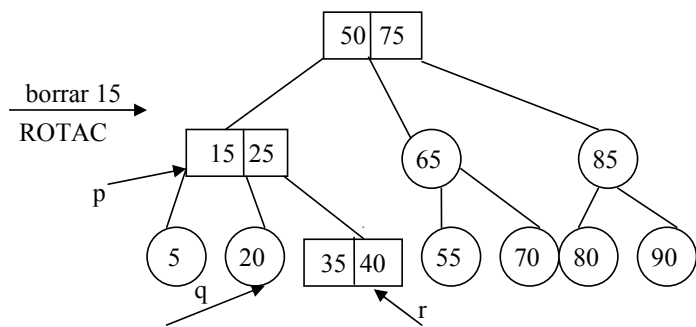
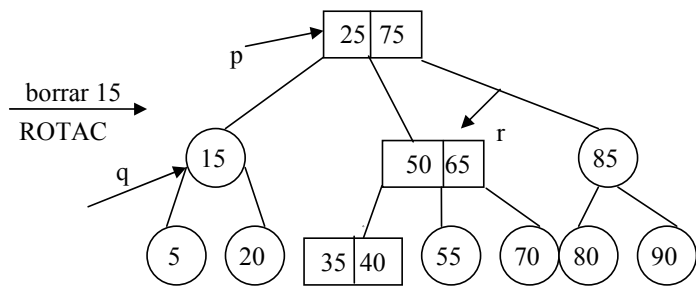
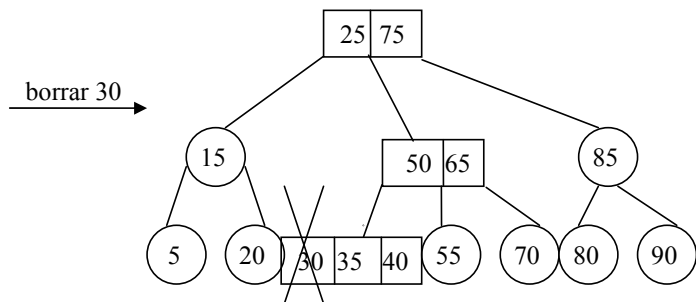
Var v: vector; i,x: natural;

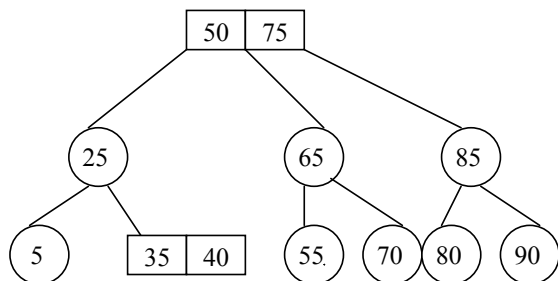
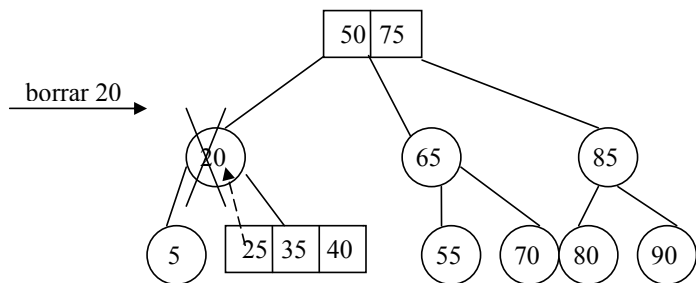
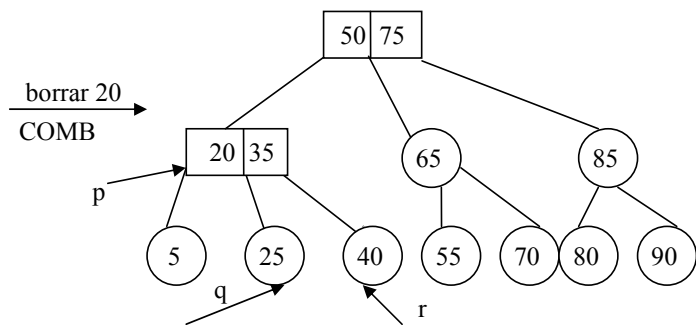
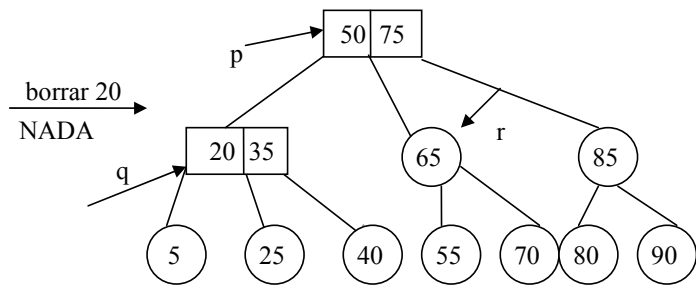
inversa(crear_vector()) = crear_vector() //caso base

inversa(asig(v,i,x)) = asig(inversa(v),100-i+1,x) //caso general

2)

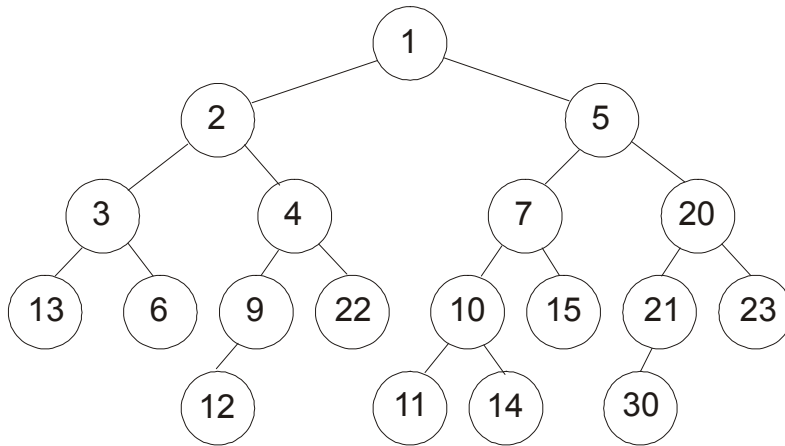




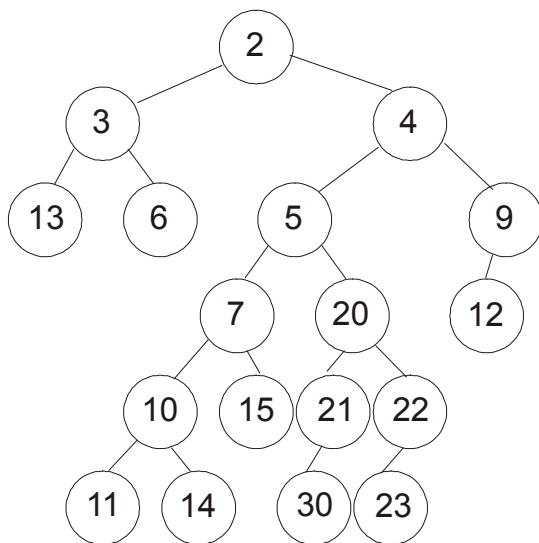


3) LEFTIST

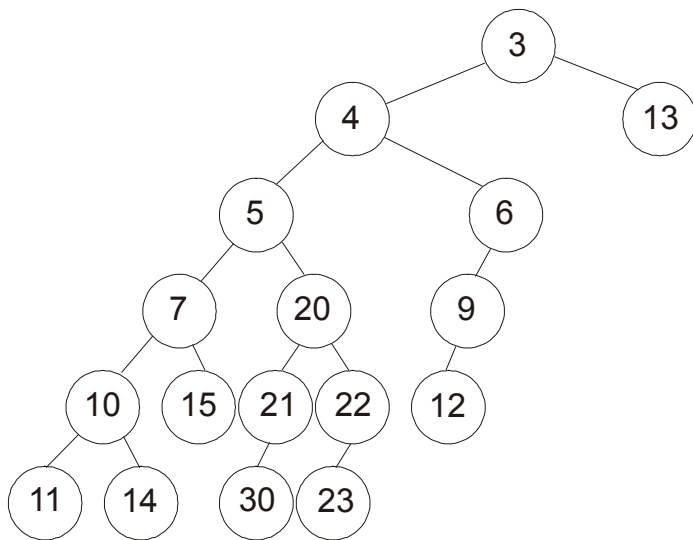
A)



B)



c)



4) COMPLEJIDAD

- Inserción y borrado en una lista enlazada no ordenada.
Inserción $O(n)$.
Borrado $O(n)$.
- Inserción y borrado en un vector ordenado circular.
Inserción $O(n)$.
Borrado $O(n)$.
- Inserción de una palabra en un trie con nodos terminales.
 $O(L)$. Siendo L la longitud de la palabra.
- Obtener mínimo de un heap mínimo.
 $O(1)$
- Insertar en un árbol binario de búsqueda enlazado.
 $O(n)$
- Borrar en un árbol leftist.
 $O(\log n)$.

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen PED junio 2008

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **15 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Las ecuaciones (vistas en clase) que permiten realizar la multiplicación de números naturales son las siguientes: $\begin{aligned} &VAR\ x, y: natural; \\ &mult(cero, x) = cero \\ &mult(x, cero) = cero \\ &mult(suc(y), x) = suma(mult(y, x), x) \end{aligned}$	<input type="checkbox"/>	<input type="checkbox"/>	1	V
En C++ y cuando se emplea composición (layering), los métodos de la clase derivada pueden acceder a la parte protegida de la clase base.	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Para el siguiente algoritmo, la complejidad sería $O(n^2)$: $\begin{aligned} &for\ (i=0; i<100; i++) \\ &\quad for\ (j=0; j<100; j++) \\ &\quad\quad if\ (v[i]<v[j])\ \ v[i]=v[j]; \\ &\quad\quad else\ v[j]=v[i]; \end{aligned}$	<input type="checkbox"/>	<input type="checkbox"/>	3	F
Dados los recorridos de preorden, postorden y niveles de un árbol binario sólo se puede reconstruir un único árbol binario.	<input type="checkbox"/>	<input type="checkbox"/>	4	F
Sabiendo que A es un árbol binario de búsqueda completo y dado su recorrido inorden 1,4,6,7,9,12,14,20,23. La secuencia 12,7,20,4,9,14,23,1,6, se corresponde con su recorrido por niveles.	<input type="checkbox"/>	<input type="checkbox"/>	5	V
Cuando se borra un elemento en un AVL habrá casos en los que no sea necesario reestructurar el árbol. Para aquellos en los que sí se necesita reestructuración, se realizará una única rotación.	<input type="checkbox"/>	<input type="checkbox"/>	6	F
Dado un árbol 2-3, si la clave a borrar x está en un nodo no hoja, x se podría sustituir por la clave anterior a x en el recorrido en inorden del árbol, y continuar con el algoritmo de borrado.	<input type="checkbox"/>	<input type="checkbox"/>	7	V
La altura h de un árbol 2-3-4 con n elementos se encuentra entre los límites $\log_4 (n+1) \geq h \geq \log_2 (n+1)$.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
Un árbol binario completo es un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	9	F
Un árbol Rojo – Negro de altura 2 puede tener todos sus enlaces de color rojo	<input type="checkbox"/>	<input type="checkbox"/>	10	V
En la inserción en un árbol B se realiza un recorrido descendente desde la raíz en el que se van dividiendo los nodos llenos que nos encontremos por el camino hasta las hojas.	<input type="checkbox"/>	<input type="checkbox"/>	11	F

Examen PED junio 2008

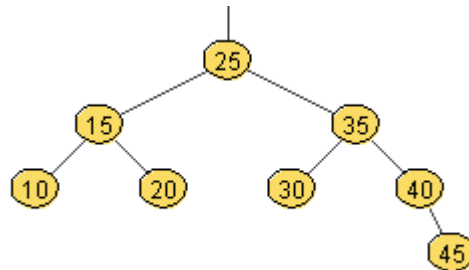
- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: *Apellidos, Nombre*.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación de notas de exámenes de teoría y práctica:** 19 de junio. **Fecha de revisión de teoría y prácticas:** 20 de junio 10:00 h. en aula LS13i de la EPS IV

- | |
|---|
| • Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas |
|---|

1. Utilizando exclusivamente las operaciones constructoras generadoras de los números Naturales define la sintaxis y la semántica de las operaciones resta y división (calcula el cociente de dos números Naturales).
NOTA: Se asume que para las dos operaciones el primer argumento es mayor que el segundo y que el cociente de la división es entero y exacto. Para la operación división se puede utilizar la operación resta.

2. Dado el siguiente árbol AVL, realiza las siguientes operaciones de forma sucesiva sobre el árbol obtenido en cada operación (criterio: sustituir por el mayor de la izquierda):

- Inserta 50, 27, 26
- Borra 20, 10, 15



3. Sea el siguiente grafo dirigido: (1,3) (1,4) (2,1) (2,4) (4,3) (4,5) (4,9) (5,3) (5,6) (6,3) (6,7) (6,9) (7,8) (10,6) (10,8) (4, 1).
- Realiza una búsqueda en profundidad comenzando por el vértice "1" y construye el bosque extendido en profundidad.
 - Etiqueta todos los arcos.
 - Realiza el recorrido en anchura (BFS) comenzando por el vértice 6.
 - ¿Este grafo tiene ciclos? Razona tu respuesta.

Nota: la lista de adyacencia se recorrerá ordenada de menor a mayor.

4.

- Decir si la siguiente afirmación es cierta. "La combinación de dos árboles leftist balanceados respecto a la altura siempre da como resultado otro árbol leftist balanceado respecto a la altura". Mostrar un ejemplo que lo demuestre.
- ¿Cuándo aumenta la altura de un árbol 2-3 al efectuar una inserción? Mostrar un ejemplo.
- Indicar de forma razonada la complejidad temporal en el peor y mejor caso de las operaciones de inserción, borrado y búsqueda en un árbol 2-3.

Examen PED junio 2008. Soluciones

1.

resta, division: natural, natural --> natural

Var x, y: natural

resta(x,cero)= x

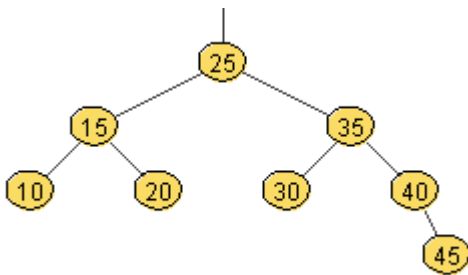
resta(suc(x), suc(y))= resta(x,y)

division(x,cero)= error_natural

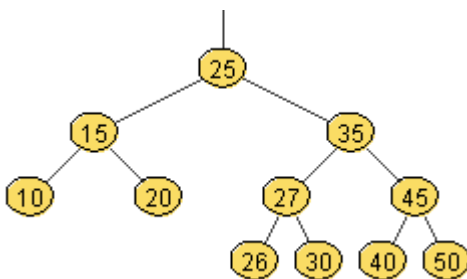
division(x,x)= suc(cero)

division(x,y)= suc(division(resta(x,y),y))

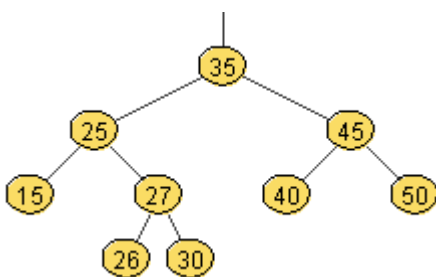
2.



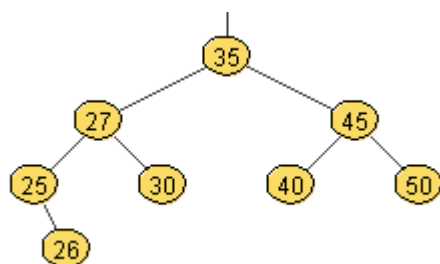
Inserta 50, 27, 26



Borra 20, 10



Borra 15



3.

a) :

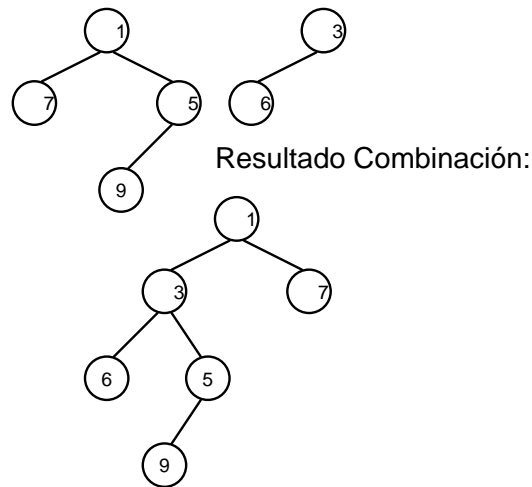
b) .

c) BFS(6): 6, 3, 7, 9, 8

d) .

4.

a) Falsa. Contraejemplo:



b) Cuando en el camino de vuelta atrás, todos los nodos por los que pasamos incluida la raíz son de grado 3.

c) Mejor caso inserción y borrado (todos los nodos del árbol son 3-nodo, por lo que la altura es menor): $\Omega(h) = \Omega(\log_3(n+1))$. Mejor caso búsqueda: $\Omega(1)$, dado que el elemento se encontrase en la raíz del árbol. Peor caso inserción, borrado y búsqueda (todos los nodos del árbol son 2-nodo, por lo que la altura es mayor): $O(h) = O(\log_2(n+1))$. Siendo n el número de nodos del árbol y h la altura del árbol.

Apellidos:

Nombre:

Convocatoria:

DNI:

Examen TAD/PED septiembre 2004

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **20 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - **El test vale un 40% de la nota de teoría.**
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
Las operaciones modificadoras de un TAD permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.	<input type="checkbox"/>	<input type="checkbox"/>	F 1.
Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca: <pre>TPosicion Tlista::Primera() { TPosicion p; p.pos = lis; return p; }</pre> <pre>class Tlista { public: ... private: Tnodo *lis; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	F 2.
En el método Primera, se invoca al constructor de Tlista.			
Sea el tipo cola definido en clase. La semántica de la operación cabeza es la siguiente: Var c:cola; x:item; cabeza(crear_cola())=error_item() si esvacía(c) entonces cabeza(encolar(c,x))=x sino cabeza(encolar(c,x))=encolar(cabeza(c),x)	<input type="checkbox"/>	<input type="checkbox"/>	F 3.
El recorrido en postorden de un árbol binario es el inverso especular del recorrido en preorden del mismo árbol	<input type="checkbox"/>	<input type="checkbox"/>	V 4.
En la operación de borrado de un ítem en un árbol AVL, si se realiza una rotación II, al menos es necesario realizar otra rotación de cualquier tipo.	<input type="checkbox"/>	<input type="checkbox"/>	F 5.
Los nodos de grado 0 de un árbol 2-3 han de estar en el mismo nivel del árbol	<input type="checkbox"/>	<input type="checkbox"/>	V 6.
Al insertar un elemento en un árbol 2-3-4 se pueden realizar una operación de DIVIDERAIZ y otra de DIVIDEHIJODE2.	<input type="checkbox"/>	<input type="checkbox"/>	V 7.
En un árbol rojo-negro ha de haber al menos un enlace rojo.	<input type="checkbox"/>	<input type="checkbox"/>	F 8.
En un árbol B tiene que haber el mismo número de nodos en el hijo izquierdo de la raíz que en el hijo derecho.	<input type="checkbox"/>	<input type="checkbox"/>	F 9.
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C: Conjunto; x: Ítem): Operación(Crear) \Leftrightarrow 0 Operación (Insertar(C, x)) \Leftrightarrow 3 + Operación(C)	<input type="checkbox"/>	<input type="checkbox"/>	V 10.
En la dispersión abierta se pueden producir colisiones entre claves sinónimas y no sinónimas	<input type="checkbox"/>	<input type="checkbox"/>	F 11.
Un recorrido en inorden de un montículo nos devolverá todos los elementos de forma ordenada.	<input type="checkbox"/>	<input type="checkbox"/>	F 12.
Un árbol 2-3 cumple las propiedades de un árbol Leftist.	<input type="checkbox"/>	<input type="checkbox"/>	F 13.
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de cruce es un grafo acíclico dirigido.	<input type="checkbox"/>	<input type="checkbox"/>	V 14.

Examen TAD/PED septiembre 2004

Normas: ♦ Tiempo para efectuar el ejercicio: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: *Apellidos, Nombre*. Cada pregunta se escribirá en folios diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible.
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación de notas de exámenes y prácticas:** 20 de septiembre. **ÚNICA fecha de revisión de exámenes:** 22 de septiembre. El lugar y la hora se publicará en el campus virtual. **EXAMEN DE PRÁCTICAS:** 20 de septiembre. El lugar y la hora se publicará en el campus virtual
- **Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas**

1. Un palíndromo es una palabra o frase que se lee igual de izquierda a derecha, que de derecha a izquierda. Proporciona la sintaxis y semántica de la operación *palíndromo*, que actúa sobre una lista cuyos elementos son caracteres y devuelve CIERTO si el contenido de la lista es un palíndromo y FALSO en caso contrario. Se considera que una lista vacía o de un solo elemento son palíndromos. Sólo se pueden emplear las operaciones de la especificación básica de listas vista en clase.
2. Se tiene una clase *TMatriz*, definida de la siguiente forma en el archivo de cabeceras (*TMatriz.h*), que está construida por *layering* a partir de *TVector*, definida en *TVector.h*:

<pre>#ifndef _TMATRIZ_ #define _TMATRIZ_ #include "tvector.h" class TMatriz { friend ostream& operator<<(ostream &, TMatriz &); public: TMatriz(int,int); //CONSTRUCTOR ~TMatriz(); //DESTRUCTOR TMatriz(const TMatriz &); //CONSTRUCTOR DE COPIA TMatriz& operator=(TMatriz &); TVector& operator[](int); private: int nfilas,ncolumnas; TVector *filas; TVector vector_error; //TRATAMIENTO DE ERRORES }; #endif</pre>	<pre>#ifndef _TVECTOR_ #define _TVECTOR_ class TVector { friend ostream& operator<<(ostream &, TVector &); public: TVector(int x=10); //CONSTRUCTOR POR DEFECTO DIMENSION 10 ~TVector(); //DESTRUCTOR TVector(const TVector &); //CONSTRUCTOR DE COPIA TVector& operator=(TVector &); int& operator[](int); private: int dimension; int* datos; int error; //TRATAMIENTO DE ERRORES }; #endif</pre>
--	--

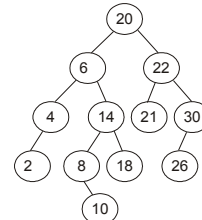
Se pide implementar los siguientes métodos para la clase *TMatriz*:

- *TMatriz(int,int); //CONSTRUCTOR*
- *TMatriz(const TMatriz &); //CONSTRUCTOR DE COPIA*
- *TVector& operator[](int);*

Comentarios:

- Utiliza por *layering* **TVector** **vector_error** para devolver el vector por defecto
- Señala dónde se emplea el *layering* en la implementación de estos 3 métodos, indicando brevemente en qué consiste.
- El rango de las FILAS de *TMatriz* es [1..i] , la de COLUMNAS (posiciones del vector) es [1..j] ; es decir, empieza en “1” y no en “0”

3. a) En un árbol AVL inicialmente vacío, insertar los siguientes elementos: 10, 3, 7, 9, 11, 15, 2, 1, 4, 5.
b) En el siguiente árbol AVL borrar los siguientes elementos: 26, 18.

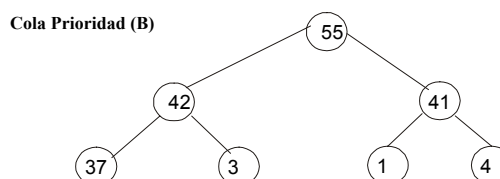
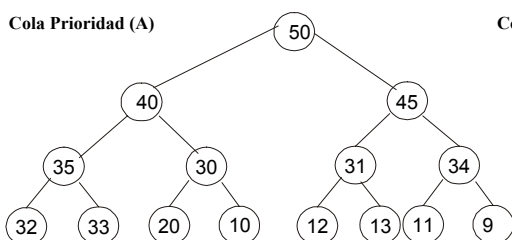


4. a) Escribir el código del método *Theap Theap::Combinar(Theap& b)* que realice la combinación de dos colas de prioridad máximas representadas mediante montículos simples. El resultado se devolverá por valor sin modificar ninguno de los dos montículos. Para ello, utilizar las clases y funciones que se indican a continuación. En caso que sea necesario se pueden utilizar funciones auxiliares. NOTA: los errores de sintaxis se valorarán negativamente.

```
Theap {
public:
    Theap(Theap&); // Constructor de copia
    void Insertar(TElemento x); // Inserta "x" en caso que no exista previamente en la cola
    TElemento Borrar(); // Devuelve el máximo de la cola, realizando su borrado
    TElemento Máximo(); // Devuelve el máximo de la cola
    Bool Vacio(); // Devuelve "true" si no contiene ningún elemento
    ... };

```

- b) Realizar la combinación de los dos montículos que aparecen a continuación (*A.Combinar(B)*) utilizando el algoritmo escrito en el apartado anterior.



Examen TAD/PED septiembre 2004. Soluciones

1)

Sintaxis:

palindromo(lista) → bool

Semántica:

VAR x: caracter; l: lista;

palindromo(crear()) = CIERTO

palindromo(incabeza(crear(), x)) = CIERTO

si x == obtener(l, ultima(l)) entonces

 palindromo(incabeza(l, x)) = palindromo(borrar(l, ultima(l)))

sino

 palindromo(incabeza(l, x)) = FALSO

2)

```
/*//////// Constructor de la matriz por layering a partir del TVector.
    IMPORTANTE: llamada al constructor del TVector para vector_error
*/
TMatriz::TMatriz(int f, int c):vector_error() // LAYERING : constructor defecto
{
    nfilas = f;
    ncolumnas = c;
    filas = new TVector[nfilas] (ncolumnas); //OJO A ESTA LLAMADA
    for ( int i=0; i<nfilas; i++ )
        for ( int j=1; j<=ncolumnas; j++ )
            filas[i][j] = 0;    ];    // LAYERING: corchete vector
}
/*****/

/*//////// Constructor de copia de la matriz.
    IMPORTANTE: llamada al constructor de copia del TVector para vector_error
*/
TMatriz::TMatriz(const TMatriz &origen):vector_error(origen.vector_error) // LAYERING: constructor copia
vector
{
    nfilas = origen.nfilas; // LAYERING: constructor copia
    ncolumnas = origen.ncolumnas; // LAYERING: constructor copia vector

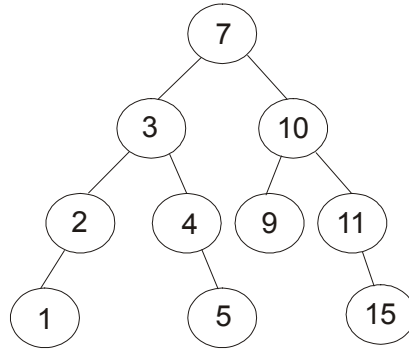
    filas = new TVector[nfilas] (ncolumnas); //OJO A ESTA LLAMADA
    for ( int i=0; i<nfilas; i++ )
        for ( int j=1; j<=ncolumnas; j++ )
            filas[i][j] = origen.filas[i][j];    // LAYERING: corchete vector
}
/*****/

TVector&
TMatriz::operator[] (int f)
{
    if(f >=1 && f <= nfilas) return (filas[f-1]); // RANGO : 1..i

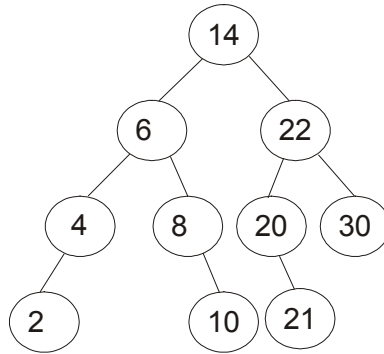
    else {
        cout<<"Fila "<<f<<" invalida para esta matriz"<<endl;
        return(vector_error);
    }
}
/*****/
```

3)

a)



b)



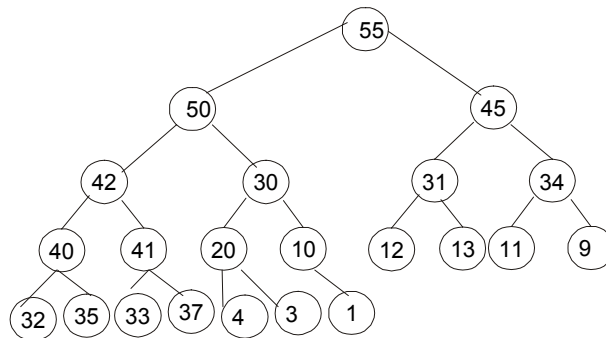
4)

a)

```

Theap
Theap::Combinar(Theap& b) {
    Theap temp(*this);
    Theap bTemp(b);
    while(!bTemp.Vacio())
        temp.Insertar(bTemp.Borrar());
    return temp;
}
  
```

b)



DNI:

Modalidad 0

	V	F		
El siguiente árbol está balanceado con respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	1.	V
La siguiente función de C++, <code>int& Incremento(int valor){valor=valor+5;return valor};</code> , devuelve el resultado por referencia.	<input type="checkbox"/>	<input type="checkbox"/>	2.	V
En las colas, las inserciones y borrados se realizan por el mismo extremo.	<input type="checkbox"/>	<input type="checkbox"/>	3.	F
La siguiente estructura es un árbol binario:	<input type="checkbox"/>	<input type="checkbox"/>	4.	F
Un árbol completo es un árbol completamente equilibrado	<input type="checkbox"/>	<input type="checkbox"/>	5.	F
Los árboles AVL son árboles balanceados con respecto a la altura de los subárboles.	<input type="checkbox"/>	<input type="checkbox"/>	6.	V
En un árbol 2-3, la diferencia en número de nodos entre los subárboles de la raíz es como mucho 1.	<input type="checkbox"/>	<input type="checkbox"/>	7.	F
Un árbol rojo-negro, en el que no hay ningún enlace rojo, es un árbol binario completo.	<input type="checkbox"/>	<input type="checkbox"/>	8.	F
Un árbol binario de búsqueda con altura 7 y 127 nodos es un árbol B con m=2	<input type="checkbox"/>	<input type="checkbox"/>	9.	V
En un árbol m-camino de búsqueda, todos los nodos excepto la raíz tienen al menos m/2 hijos.	<input type="checkbox"/>	<input type="checkbox"/>	10.	F
En la dispersión cerrada se pueden producir colisiones entre claves no sinónimas	<input type="checkbox"/>	<input type="checkbox"/>	11.	V
En un montículo doble, un elemento “j” del montículo máximo es el simétrico de un único elemento “i” del montículo mínimo.	<input type="checkbox"/>	<input type="checkbox"/>	12.	F
Un árbol Rojo-Negro cumple las propiedades de un árbol Leftist.	<input type="checkbox"/>	<input type="checkbox"/>	13.	F
Al representar un grafo no dirigido con una matriz de adyacencia, su diagonal principal (casillas i,i) siempre tendrá valores Falso.	<input type="checkbox"/>	<input type="checkbox"/>	14.	V

Examen TAD/PED septiembre 2005

Normas: ♦ Tiempo para efectuar el ejercicio: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **Apellidos, Nombre. LAS PREGUNTAS 1 Y 2 SE ENTREGARÁN JUNTAS, IGUALMENTE LA 3 y 4.**
- Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
- Publicación de notas de exámenes:** 13 de septiembre. **Fecha de revisión de exámenes:** 19 septiembre, de 11:30 a 13:00 h. en el laboratorio OIN01 de la EPSA IV.

• Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Completa en esta misma hoja las ecuaciones que aparecen a continuación y que expresan el comportamiento de las operaciones de: a) *obtener* en una lista de acceso por posición, b) *resta* en el conjunto de los números Naturales en el que sólo existen las operaciones *cero*: \rightarrow *natural* y la operación *suc*: *natural* \rightarrow *natural* (devuelve el sucesor de un número Natural). Se asume que el primer operando de la *resta* es siempre mayor o igual que el segundo.

a) *obtener*(lista,posicion) \rightarrow item

obtener(crear(),p) =

si p = primera (inscabeza(l₁,x)) entonces *obtener*(inscabeza(l₁,x),p) =

sino *obtener*(inscabeza(l₁,x),p) =

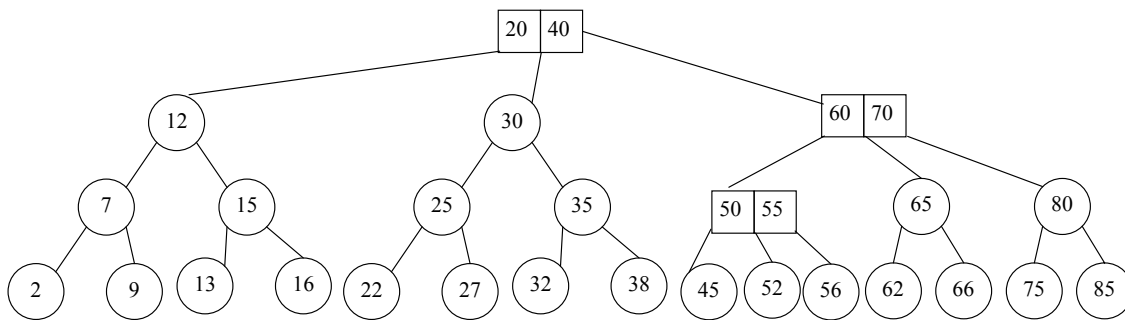
b) *resta*(natural,natural) \rightarrow natural

resta(,) =

resta(,) =

Donde: l₁ ∈ lista x ∈ item p ∈ posicion a, b ∈ natural

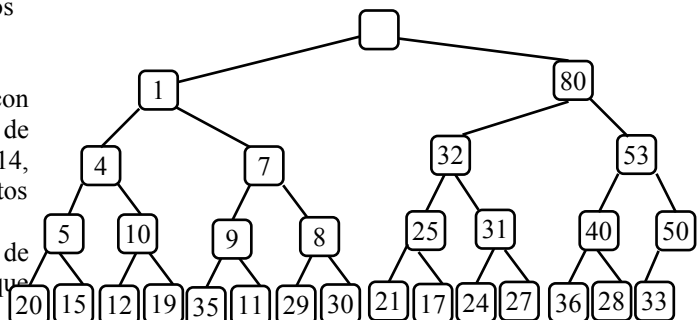
2. Dado el siguiente árbol 2-3 borrar los elementos 40, 30 y 12. (Criterios: (1) si el nodo tiene dos hijos hay que sustituir por el mayor de la izquierda, (2) si el 2-nodo tiene dos hermanos, consultar el hermano de la derecha). Realiza 1 gráfico para el borrado de cada elemento indicando las transformaciones realizadas.



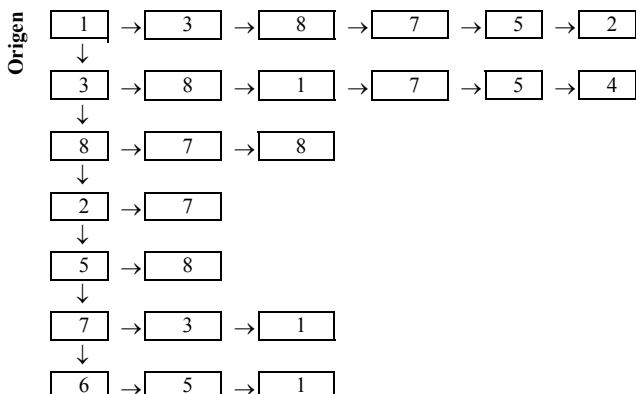
3. Dado el siguiente montículo doble borrar los elementos mínimo y máximo de forma sucesiva.

4. a) Insertar en una tabla de dispersión cerrada vacía, con estrategia de redistribución con una segunda función Hash, de tamaño B=11, la siguiente secuencia de elementos: 23, 14, 10, 15, 3, 5, 7, 8, 36, 47 y 4. Detallar la secuencia de intentos

b) Dado el grafo dirigido representado por la lista de listas que se muestra a continuación, obtener el bosque extendido en profundidad que parte del vértice 1, con su clasificación de arcos. Para recorrer las listas de adyacencia, seguir el orden de izquierda a derecha de cada lista.



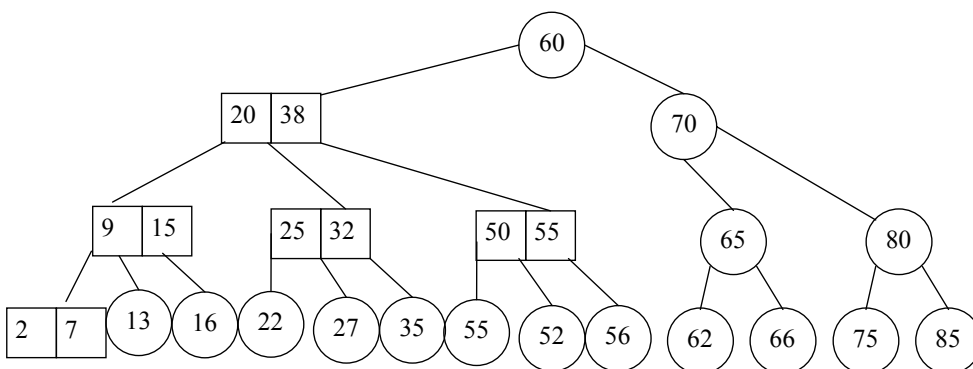
Destino



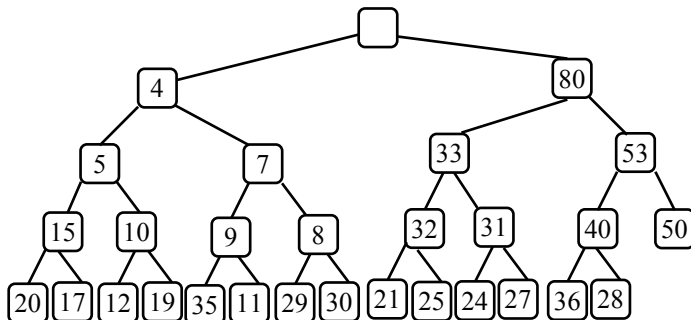
Examen TAD/PED septiembre 2005. Soluciones

1. a) obtener(lista,posicion) → item
 obtener(crear(),p) = **error_item**.....
 si p==primera (inscabeza(l₁,x)) entonces obtener(inscabeza(l₁,x),p)= **x**
 sino obtener(inscabeza(l₁,x),p)= **obtener(l₁,p)**.....
 b) resta(natural,natural) → natural
 resta (**a** , **cero**) = **a**
 resta (**suc(a)** , **suc(b)**) = **resta(a,b)**.....
2. BORRADO DEL 40: 2 COMBINACIONES Y 1 ROTACIÓN
 BORRADO DEL 30: 1 ROTACIÓN
 BORRADO DEL 12: 3 COMBINACIONES

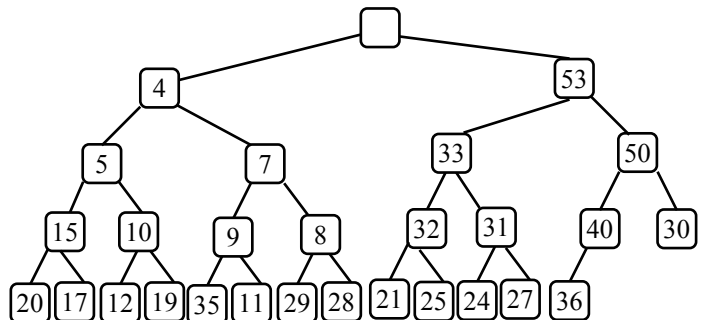
RESULTADO FINAL:



3. a) Borrado del mínimo:



b) Borrado del máximo:



4. a) $k(x) = (x \text{ MOD } (B-1)) + 1$ $h_i(x) = (H(x) + k(x) \cdot i) \text{ MOD } B$

0	1	2	3	4	5	6	7	8	9	10
47	23	4	14	15	5	36	3	8	7	10

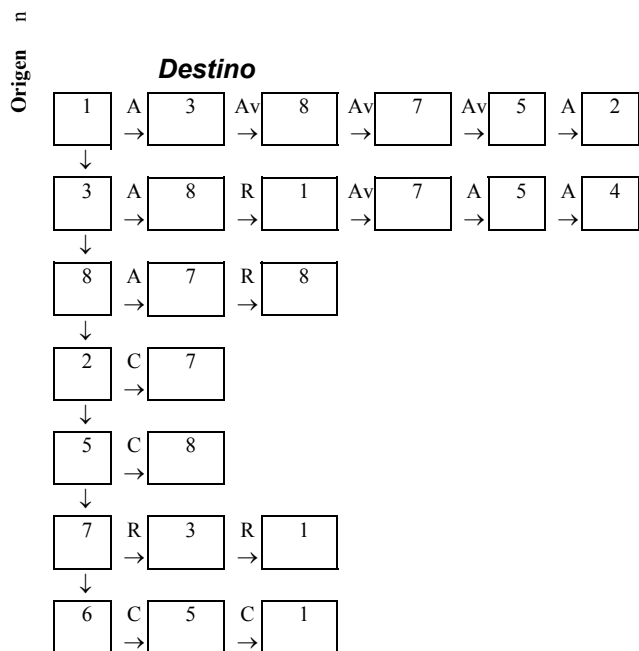
4. b)

Recorrido en profundidad: 1, 3, 8, 7, 5, 4, 2, 6

Arcos de avance: <3, 7>, <1, 8>, <1, 7>, <1, 5>

Arcos de cruce: <5, 8>, <2, 7>, <6, 5>, <6, 1>

Arcos de retroceso: <7, 3>, <7, 1>, <3, 1>, <8, 8> **Arcos del árbol:** los restantes



Apellidos:

Nombre:

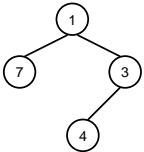
Convocatoria:

DNI:

Examen TAD/PED septiembre 2006

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **18 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - El test vale un 40% de la nota de teoría.
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
Las operaciones constructoras generadoras de un tipo permiten obtener cualquier valor de dicho tipo.	<input type="checkbox"/>	<input type="checkbox"/>	1. V
En C++, si no se ha implementado la sobrecarga del operador asignación, se invoca automáticamente al constructor de copia.	<input type="checkbox"/>	<input type="checkbox"/>	2. F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/>	3. V
La semántica de la operación nodos del tipo <i>arbin</i> vista en clase es la siguiente: $VAR i, d: arbin; x: item;$ $nodos(crea_arbin()) = 0$ $nodos(enraizar(i, x, d)) = nodos(i) + nodos(d)$	<input type="checkbox"/>	<input type="checkbox"/>	4. F
Se puede reconstruir un único árbol binario cualquiera teniendo sus recorridos en preorden y postorden.	<input type="checkbox"/>	<input type="checkbox"/>	5. F
La semántica de la operación <i>recu</i> vista en clase es la siguiente: $VAR v: vector; i, j: int; x: item;$ $recu(crear_vector(), i) = error_item()$ $recu(asig(v, i, x), j)$ si $(i == j)$ entonces x sino FALSO fsi	<input type="checkbox"/>	<input type="checkbox"/>	6. F
En un árbol AVL cuyo factor de equilibrio es -2, al insertar un elemento en la rama derecha, el árbol vuelve al estado de equilibrio.	<input type="checkbox"/>	<input type="checkbox"/>	7.
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 $O(\log_2 n)$	<input type="checkbox"/>	<input type="checkbox"/>	8. V
Un árbol binario de búsqueda lleno de altura 4 es un árbol 2-3-4, pero no se puede conseguir a partir de un árbol inicialmente vacío y utilizando las operaciones de inserción y borrado de un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	9. V
Un grafo no dirigido puede tener aristas que empiecen y acaben en el mismo vértice.	<input type="checkbox"/>	<input type="checkbox"/>	10. F
El siguiente árbol es leftist mínimo: 	<input type="checkbox"/>	<input type="checkbox"/>	11. V
Un trie cumple las propiedades de un árbol general.	<input type="checkbox"/>	<input type="checkbox"/>	12. V

Examen PED septiembre 2006

- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** 18 de septiembre. **Revisión exámenes TEORÍA:** 19 de septiembre de 9:30 a 10:30 en aula LS14I del sótano de la EPS IV
- Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Utilizando exclusivamente las operaciones constructoras generadoras del tipo conjunto, definid la sintaxis y la semántica de la operación *subconjunto_impares* que se aplica sobre un conjunto dado de números naturales y devuelve el subconjunto formado por los números impares que existen en el conjunto original.

Ej: $C = \{1, 8, 10, 3, 12\}$

$\text{subconjunto_impares}(C) = \{1, 3\}$

Nota: Se asume que está definida la operación MOD para calcular el resto de la división entera.

2. Dada la siguiente declaración de la clase *TLista* que representa una lista ordenada (de menor a mayor) doblemente enlazada de números enteros donde no se permiten repetidos, escribe el código del método *bool Insertar(int)*, que devuelve cierto si el número se puede insertar y falso en caso contrario. Se proporciona el código del constructor y destructor de la clase *TNodo*.

Nota: se tiene que escribir el código de todos los métodos auxiliares que se empleen.

```
class TLista {
public:
    TLista();
    ...
    ~TLista();
    ...
    bool Insertar(int);
    ...
private:
    TNodo *primero, *ultimo;
};

class TNodo {
public:
    TNodo();
    ...
    ~TNodo();
    ...
private:
    int etiqueta;
    TNodo *anterior, *siguiente;
};

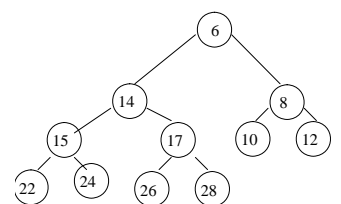
TLista::TLista() {
    primero = NULL;
    ultimo = NULL;
}

TNodo::TNodo() {
    etiqueta = 0;
    anterior = NULL;
    siguiente = NULL;
}

TNodo::~TNodo() {
    etiqueta = 0;
    anterior = NULL;
    siguiente = NULL;
}
```

3. Dado el siguiente árbol Leftist mínimo (izquierdista mínimo):

- Insertar los siguientes elementos en el orden que se indica: 13, 16, 20, 11, 4, 2, 30, 1
- Sobre el Leftist resultante, realizar el borrado de los 3 ítems mínimos.



4. Dado el **grafo no dirigido** representado por la siguiente diagonal superior de una matriz de adyacencia, tal y como se ha visto en clase:

	1	2	3	4	5	6	7	8
1					•			•
2							•	•
3					•			
4					•	•	•	•
5								•
6								
7								•
8								

- Realizar el recorrido en profundidad y en anchura empezando por 1.
- Obtener el bosque extendido en profundidad y clasificación de arcos.
- Obtener razonadamente la complejidad en el peor caso de la operación *obtenerAdyacencia(v)* utilizando la representación interna del grafo no dirigido mostrado en este ejercicio.

NOTA: Se seguirá el criterio de recorrer la adyacencia de cada vértice ordenada de menor a mayor vértice.

Examen PED septiembre 2006. Soluciones

1.

Sintaxis

subconjunto_impares: conjunto \rightarrow conjunto

Semántica

```
Var C: conjunto; x: natural;
subconjunto_impares(crear_conjunto()) = crear_conjunto()
subconjunto_impares(Insertar(C,x))=
    si (x MOD 2 ==1) entonces Insertar(subconjunto_impares(C),x)
    si no subconjunto_impares (C)
```

2.

```
bool
TLista::Insertar(int item) {
    if(primeros == NULL)
    {
        primeros = new TNode;
        if(primeros == NULL)
            return false;

        ultimo = primeros;
        primeros->etiqueta = item;
        return true;
    }
    else
    {
        TNode *aux, *nuevo;

        if(item < primeros->etiqueta)
        {
            nuevo = new TNode;
            if(nuevo == NULL)
                return false;

            nuevo->etiqueta = item;
            nuevo->siguiente = primeros;
            primeros->anterior = nuevo;
            primeros = nuevo;
            return true;
        }

        aux = primeros;
        while(aux)
        {
            if(item < aux->etiqueta)
            {
                nuevo = new TNode;
                if(nuevo == NULL)
                    return false;

                nuevo->etiqueta = item;
                nuevo->anterior = aux->anterior;
                nuevo->siguiente = aux;
                aux->anterior->siguiente = nuevo;
                aux->anterior = nuevo;
                return true;
            }
            else if(item == aux->etiqueta)
                return false;
            aux = aux->siguiente;
        }

        nuevo = new TNode;
        if(nuevo == NULL)
            return false;

        nuevo->etiqueta = item;
        nuevo->anterior = ultimo;
```

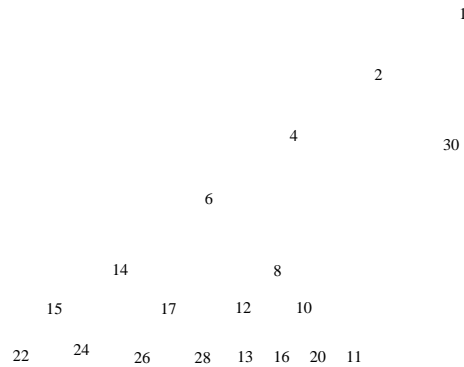


```

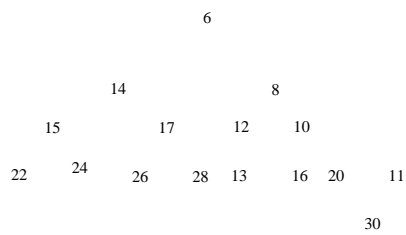
    ultimo->siguiente = nuevo;
    ultimo = nuevo;
    return true;
}
}

```

3. a)



b)



4. a) DFS: 1, 5, 3, 4, 6, 7, 2, 8
 BFS: 1, 5, 8, 3, 4, 2, 7, 6
 b)

	1	2	3	4	5	6	7	8
1					A			R
2							A	A
3					A			
4					A	A	A	R
5								R
6								
7								R
8								

c) $O(n)$, con n el número de vértices del grafo, ya que el peor caso sería el de *obtenerAdyacencia(8)* en el que nos obligaría a recorrer toda la columna del 8, o bien el caso de *obtenerAdyacencia(1)* en el que tendríamos que recorrer toda la fila del 1.