Examen de Estructuras de Datos. Grado en Ing. Informática. Febrero 2012.

1. (1 punto) Rellenar la siguiente tabla con la eficiencia de las operaciones de las filas en las estructuras de datos de las columnas. El tipo de dato subyacente es siempre **int**.

	vector ord.	stack	priority_queue	list	avl	tabla hash
Buscar elemento						
Encontrar máximo						
Num. elem. en [a,b] $a < b$						
Insertar elemento						
Imprimir						

2. (1.5 puntos) Se quiere construir el tipo de dato `vectorDisperso` de string, que se caracteriza porque la mayoría de los elementos toman un mismo valor (que llamaremos valor por defecto). Para representar dicho vector es más eficiente almacenar solo los elementos distintos al valor por defecto, por lo que se propone la siguiente representación:

```
class vectorDisperso {
  ....
  private:
     map<int,string> M; // map donde se alojan los elementos no nulos.
     string v_def;  // valor por defecto
};
```

donde, para un `vectorDisperso V`, el elemento de la posición i-ésima del vector, V[i], sería:
$V[i] = \begin{cases} M[i] & si M.find(i) != M.end(). \\ v\_def & \text{en caso contrario.} \end{cases}$

(a) Dar una especificación (sintaxis y semantica) de los 5 métodos que consideres más relevantes de ese nuevo TDA. Analizar la representación propuesta y determinar la FA y el IR.
(b) Implementar el método que cambie el valor por defecto del vector (hay que tener cuidado con los elementos del vector disperso anterior cuyo valor anterior fuese nv).

```
void vectorDisperso::cambiar_valor_defecto( const string & nv)
```

3. (a) (2 puntos) Usando el TDA list<T>, construir una función **template <class T> void** dividir-lista (**list<T> &** entrada, **T** k) que agrupe en la primera parte de la lista los elementos menores que k y en la segunda los mayores o iguales. Han de usarse iteradores, no se permite usar ninguna estructura auxiliar, no puede modificarse el tamaño de la lista y la función ha de ser O(n). P.ej. Si es una list<int> **entrada**={1,3,4,14,11,9,7,16,25,19,7,8,9 } y **k**=8, entonces la lista quedaría p.ej. como: **entrada** = $\{\underbrace{1,3,4,7,7}, \underbrace{9,11,16,25,19,14,8,9}\}$.

(b) (1.5 puntos) Sea A un árbol binario con n nodos. Se define el ancestro común más cercano (AMC) entre 2 nodos v y w como el ancestro de mayor profundidad que tiene tanto a v como a w como descendientes (se entiende como caso extremo que un nodo es descendiente de si mismo). Diseñar una función que tenga como entrada un árbol binario de enteros y dos nodos v y w y como salida el AMC de v y w.

4. (a) (1 punto) Construir un AVL (especificando los pasos seguidos) a partir de las claves {100, 29, 71, 82, 48, 39, 101, 22}. Indicar cuando sea necesario el tipo de rotación que se usa para equilibrar.
   (b) (1 punto) Insertar las claves {10, 39, 6, 32, 49, 18, 41, 37} en una Tabla Hash cerrada de tamaño 11, usando como función hash h(k)=(2k+5) % 11 y para resolver las colisiones rehashing doble usando como función hash secundaria h_0(k) = 7- (k % 7). ¿Cual es el rendimiento de la tabla?
   (c) (1 punto) Usando la clase **set** de la STL, construir una función que divida un conjunto de enteros *c* en dos subconjuntos *cpar* y *cimpar* que contienen respectivamente los elementos pares e impares de *c* y que devuelva **true** si el número de elementos de *cpar* es mayor que el de *cimpar* y **false** en caso contrario.

```
/**
  @brief Determina si un conjunto de enteros tiene mas elementos pares que impares
  @brief estableciendo su particion en dos subconjuntos
  @param c: conjunto de entrada
  @param cpar: subconjunto conteniendo los elementos pares
  @param cimpar: subconjunto conteniendo los elementos impares
  @return true si #cpar > #cimpar, false en caso contrario
*/
bool masparesqueimpares(const set<int> & c, set<int> & cpar, set<int> & cimpar)
```

5. (2 puntos) Dado un árbol binario de enteros, se dice que está sesgado a la izquierda si para cada nodo, se satisface que la etiqueta de su hijo izquierda es menor que la de su hijo derecha (en caso de tener un solo hijo, este ha de situarse necesariamente a la izquierda).

   Se pide

   - (a) Implementar un método que compruebe si un Arbol binario A esta sesgado hacia la izquierda.

   - (b) Implementar un método que transforme un Arbol binario en un Arbol sesgado hacia la izquierda. La transformación debe preservar que si un nodo v es descendiente de otro w en A, también lo debe ser en el Arbol transformado, por lo que no es valido hacer un intercambio de las etiquetas de los nodos.

Notas

- En la pregunta 4, los apartados 4(a) y 4(b) son excluyentes. Solo se podrá contestar a uno de ellos.
- El tiempo para realizar el examen es de 3 horas

Universidad de Granada
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
ETSII Informática y de Telecomunicación, C/Periodista Daniel Saucedo Aranda s/n - 18071 - Granada (España)

Estructuras de Datos

Curso 2012–2013. Convocatoria de Febrero

Grado en Ing.Informática y Doble Grado en Ing.Informática y Matemáticas

1. (1 punto) Usando la notación O, determinar la eficiencia de la siguiente función (n es una potencia de 2):

```
int eficexamen(bool existe)
{
     int sum2=0; int k,j,n;
     if (existe)
           for(k=1; k<=n; k*=2)
                for(j=1; j<=k; j++)
                      sum2++;
     else
           for(k=1; k<=n; k*=2)
             for(j=1; j<=n; j++)
                      sum2++;
   return sum2;
}
```

2. (1 punto) Rellenar la siguiente tabla con la eficiencia de las operaciones de las filas en las estructuras de datos de las columnas.

	Vector ordenado	Lista ordenada	Pila	ABB	AVL	Tabla Hash
Buscar						
Insertar						
Borrar						
Imprimir ordenadamente						

3. (2 puntos) Una empresa mantiene datos sobre personas (DNI, apellidos, nombre, domicilio y teléfono, todos de tipo string) y necesita poder hacer búsquedas sobre ellos tanto por DNI como por apellidos. Define una representación eficiente (discutiendo las alternativas) para manejar los datos de esta empresa e implementa las operaciones de inserción y de borrado de una persona en dicha estructura. Las cabeceras serian:

   - `void datos::insertar(const persona &p)`
   - `void datos::borrar(const persona &p)`

4. (2 puntos) Dadas dos listas de intervalos cerrados [ini, fin], L1 y L2, donde para cada lista los intervalos se encuentra ordenados por orden de tiempo de inicio satisfaciendo que si un intervalo it1 está antes que otro it2 en la lista entonces it1.fin $<$ it2.ini.

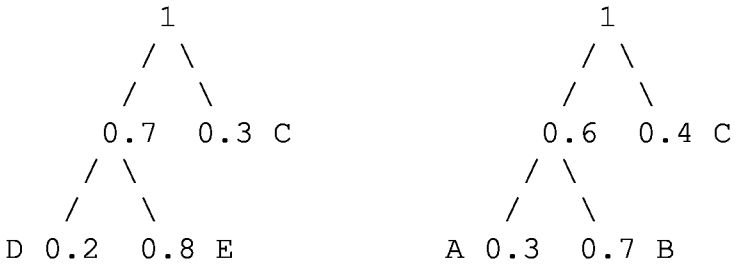
   Diseña un método que permita seleccionar un (sub)intervalo de L1 y lo pase a L2. En este proceso podría ser necesario el dividir un intervalo de L1 así como fusionar intervalos en L2 cuando ocurran solapamientos.

   ```
   typedef pair<int,int> intervalo;
   ```

```
bool Extraer(list<intervalo> & L1, intervalo x, list<intervalo> & L2);
```
Devuelve true si ha sido posible realizar la extracción (x debe pertenecer a un único intervalo de L1).

Por ejemplo, si L1 = [1,7], [10,14], [18,20], [25,26] ; L2 = [0,1], [14,16], [20,23] y x = [12,14] entonces las listas quedarán como L1 = [1,7], [10,11], [18,20], [25,26] y L2 = [0,1], [12,16], [20,23] . De igual forma, si L1 = [1,7], [10,22], [25,26] ; L2 = [0,1], [14,16], [20,23] y x = [12,20] entonces las listas quedarán como L1 = [1,7], [10,11], [21,22], [25,26] y L2 = [0,1], [12,23] .

5. (2 puntos) Un árbol de sucesos es un árbol binario donde cada nodo tiene asociada una etiqueta con un valor real en el intervalo [0,1]. Cada nodo que no es hoja cumple la propiedad de que la suma de los valores de las etiquetas de sus hijos es 1. Un suceso es una hoja y la probabilidad de que éste ocurra viene determinada por el producto de los valores de las etiquetas de los nodos que se encuentran en el camino que parte de la raiz y acaba en dicha hoja. Se dice que un suceso es probable si la probabilidad de que ocurra es mayor que 0.5. Usando el TDA árbol binario:
(a) Diseñar una función que compruebe si un árbol binario A es un árbol de sucesos. Su prototipo será `bool check_rep (const bintree<float> &A)`
(b) Diseñar una función que indique si existe algún suceso probable en el árbol de probabilidades A. Su prototipo será:
`bool probable (const bintree<float> & A)`
Ejemplo: En el árbol de probabilidades de la derecha no existe un suceso probable porque los tres sucesos tienen probabilidad inferior a 0.5 (A: 1.0*0.6*0.3=0.18; B: 1.0*0.6*0.7=0.42; C: 1.0*0.4=0.4). En cambio, en el árbol de la izquierda hay un suceso probable: E, porque 1.0*0.7*0.8=0.56



6. (2 puntos) Supongamos que en una Tabla Hash abierta hacemos uso en cada cubeta de árboles AVL en lugar de listas.

   - ¿Cual es la eficiencia (en el caso peor) de la función buscar?
   - Mostrar la tabla resultante de insertar los elementos {16, 72, 826, 1016, 12, 42, 623, 22, 32}.

x	16	72	826	1016	12	42	623	22	32
h(x)	6	2	6	6	2	2	3	2	2

   - Construir un APO a partir del anterior conjunto de claves y a continuación elimina el elemento más pequeño.

Notas

- **Tiempo para realizar el examen**: 3 horas.

Universidad de Granada

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIA

ETSII Informática y de Telecomunicación, C/ Periodista Daniel Saucedo Aranda s/n- 18071- Granada (España)

Estructuras de Datos

Curso 2012-2013. Convocatoria de Septiembre

Grado en Ingeniería Informática.

1. (2 puntos) Un videoclub está desarrollando una aplicación que permita un acceso rápido a las películas de las que disponen. No se tiene claro qué estructura de datos elegir para almacenar los registros (compuestos por código, título, año...). La **búsqueda** se hace por el título de la película . Para la **inserción** y **borrado** de películas, se usa un código único asociado a cada película. Además se necesita una función para **imprimir** de forma ordenada todas las películas. Analizar la eficiencia de las 4 operaciones si se usa como estructura de datos: 1) **vector ordenado**, 2) **lista ordenada**, 3) **ABB**, 4) **AVL** y 5) **Tablas Hash**. En función del resultado obtenido en el análisis de eficiencia realizado, decidir que estructura de datos resuelve mejor el problema.
2. (2 puntos) Para gestionar un documento se utiliza un TDA Documento. Este TDA tiene en su representación una tabla hash, en la que cada palabra del documento tiene asociada una lista ordenada con las posiciones en las que aparece la palabra en el mismo. Implementa una función

   int Documento::min_distancia(string pal1, string pal2)

   que devuelva la distancia mínima en la que aparecen las palabras *pal1* y *pal2* en el documento. Para la representación de la tabla hash se usa hash abierto.
3. (2 puntos) Dada una lista de enteros con elementos repetidos, diseñar (usando el TDA lista) una función que construya a partir de ella una lista ordenada de listas, de forma que en la lista resultado los elementos iguales se agrupen en la misma sublista. Por ejemplo si ***entrada =*{1,3,4,5,6,3,2,1,4,5,5,1,1,7}** entonces ***salida=*{ {1,1,1,1},{2},{3,3},{4,4},{5,5,5},{6},{7} }**
4. (2 puntos) Un árbol binario se dice que es ***k-balanceado*** si, para cada nodo, la diferencia entre el número de nodos en el subárbol izquierdo y derecho no es mayor que k. Diseñar (usando el TDA Arbol Binario) una función que permita determinar si un árbol es k balanceado.
5. (2 puntos) Disponemos del TDA Matriz de enteros (se almacenan los datos por filas) y se quiere definir un iterador que itere por columnas sobre los elementos pares de la matriz. Para ello hay que implementar los operadores ++ y *, así como las funciones begin() y end() en la clase Matriz. Por ejemplo si la matriz M tiene los siguientes datos :

   $$\begin{pmatrix} 5 & 4 & 3 \\ 2 & 1 & 2 \\ 9 & 0 & 2 \\ 8 & 9 & 1 \end{pmatrix}$$

   Ejecutando el siguiente código

```
Matriz M;
....
Matriz:: iterator it;
for (it=M.begin(); it!=M.end();++it)
  cout<<*it;
```

   se imprimirá sobre la salida estándar: **2 8 4 0 2 2**

Tiempo para realizar el examen:3 horas

Universidad de Granada

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIA

ETSII Informática y de Telecomunicación, C/ Periodista Daniel Saucedo Aranda s/n- 18071- Granada (España)

Estructuras de Datos
Curso 2013-2014. Convocatoria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (1 punto) Dado el *TDA Montón de Cartas*, con N cartas de la baraja española, y con las siguientes operaciones:
   - **Barajar:** Dispone las N cartas en orden aleatorio
   - **CogerCarta:** Devuelve la carta en el tope del montón
   - **EliminarCarta:** Elimina la carta en el tope del montón
   - **InsertarCarta**: Inserta una carta al final del montón

   a) Deducir la eficiencia de las cuatro funciones suponiendo que representamos el *TDA Montón de Cartas como: 1)Vector, 2)Lista, 3) Pila y 4) Cola*
   b) *Implementar la función Barajar suponiendo que el TDA Montón de Cartas se representa como: 1)Vector, 2)Lista, 3) Pila y 4) Cola. Nota: Se puede usar si es necesario objetos contenedores auxiliares para su implementación.*
2. (1 punto) Dados dos multiset con elementos enteros, implementar la función:
   multiset<int> multi_interseccion (const multiset<int> & m1, const multiset<int> &m2)
   que calcula la intersección de dos multiset: *elementos comunes en los dos multiset repetidos tantas veces como aparezcan en el multiset con menor número de apariciones del elemento.*
   Por ejemplo siendo m1={2,2,3,3} y m2={1,2,3,3,3,4} entonces m1∩m2={2,3,3} ó si m1={2,2,2,3,3} y m2={1,2,2,2,2,3,3,3,4} entonces m1∩m2={2,2,2,3,3}
3. (2 puntos) Suponed que tenemos el T.D.A. Tabla Hash abierta (unordered_set) (class TH), en la que la resolución de colisiones se hace utilizando para cada cubo una lista.

```
1. #include <vector>
2. #include <list>
3. using namespace std;
4. class TH{
5. private:
6.   struct info{
7.     int key;//clave
8.     int di;//dirección
9.   };
10. vector <list<info> > data;
11. int fhash(int k)const;
12. bool recolocar()const;
13. public:
14.        ...
15./*  k  la  clave,  d  la  direccion
   asociada para esa clave*/
16.void insertar(int k, int d);
17.   ...
18. class iterator{
19. private:
20. list<info>::iterator it_cub;
21. vector<list<info> >::iterator it;
22.             ...
23. }
24. iterator begin();
25. iterator end();
26.             ...
27.};
```

   a) Implementar la función ***insertar*** suponiendo que tenemos implementada la función hash (***fhash***). Después de añadir la nueva clave k y su dirección asociada d, la función ***insertar*** debe comprobar si es necesario redimensionar la tabla hash. Para ello se supone que tenemos implementada la función ***recolocar***. Esta función devuelve verdadero en el caso que sea necesario pasar todos los datos a un nueva tabla de mayor tamaño, y falso en caso contrario. El tamaño de la nueva tabla tiene que ser el primo más cercano a 2M por exceso, siendo M el tamaño original.
   b) Implementar la ***clase iterator*** (un iterador sobre todos los elementos de la tabla hash) de la

tabla hash, así como las funciones ***begin*** y ***end*** de la clase TH.

4. (2 puntos)
   a) Dado un árbol binario de enteros (positivos y negativos) implementar una función que obtenga el número de caminos, en los que la suma de las etiquetas de los nodos que los componen sumen exactamente **k.**

      int NumeroCaminos(bintree<int> & ab, int k)

   b) Construir el AVL y el APO que resultan de insertar (en ese orden) los elementos del conjunto de enteros {45,23,12,20,15,22,24,55,52}.
5. (1 punto) Describid las similitudes y diferencias (razonando la respuesta) en el funcionamiento del método básico de inserción de un elemento el los siguientes tipos de datos abstractos : 1) vector, 2) list, 3) map, 4) set, 5) priority_queue, 6) tabla hash abierta (unordered_set).

Tiempo: 3 horas

Universidad de Granada

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIA

ETSII Informática y de Telecomunicación, C/ Periodista Daniel Saucedo Aranda s/n- 18071- Granada (España)

Estructuras de Datos
Curso 2013-2014. Convocatoria de Septiembre
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (1.5 puntos) Se desea diseñar un TDA, que llamaremos **colec-ord** que representa a una colección ordenada (de menor a mayor) de enteros y donde las únicas operaciones permitidas son **insertar**, **borrar-máximo**, **recuperar** y **cuantos-dentro-de-rango**. La operaciones borrar-máximo y recuperar, respectivamente borran el elemento máximo y recuperan un entero en la colección, y la operación cuantos-dentro-de-rango tiene como parámetros un rango de valores enteros [a,b] (a<b) y devuelve un valor entero indicando **cuantos** de los enteros de la colección son mayores o iguales que **a** y menores o iguales que **b**. Analizar la eficiencia de las operaciones si se usa (a) vector de la STL (b) lista de la STL (c) un ABB, (d) un AVL. ¿Qué implementación escogerías para el TDA?

2. (2.5 puntos) Se desea construir un traductor de un idioma origen a un idioma destino. Una palabra en el idioma origen puede tener más de un traducción en el idioma destino.
   - Dar una representación para el TDA Traductor
   - Implementar la función **insertar** que añade una palabra del idioma origen junto con las traducciones en el idioma destino.
   - Implementar la función c**onsultar** que obtiene las traducciones de una palabra en el idioma destino.
   - Implementar la clase **iterador** dentro de la clase Traductor para poder iterar sobre todas las palabras.

3. (2 puntos) Construir una función que permita "postintercalar" dos listas (intercalar alternativamente todos los nodos que las integran de final a principio), con los siguientes requisitos:
   (a) Hay que comprobar que las dos listas no son vacías
   (b) Se empieza por el primer nodo de la primera lista
   (c) La segunda lista contendrá el resultado final y la primera quedará vacía.
   (d) Si una de ellas tiene un menor número de nodos que la otra, el exceso de nodos se incorporará a la lista resultante.

Ejemplo 1:
Antes de invocar al metodo
Primera lista : (100,200)
Segunda lista: (1,2,3,4,5,6)
Despues de invocar al metodo
Lista segunda: (1,2,3,4,200,5,100,6)
Lista primera: vacia

Ejemplo 2:
Antes de invocar al metodo
Primera lista : (x,y,z)
Segunda lista: (a,b,c,d,e,f)
Despues de invocar al metodo
Lista segunda: (a,b,c,z,d,y,e,x,f)
Lista primera: vacia

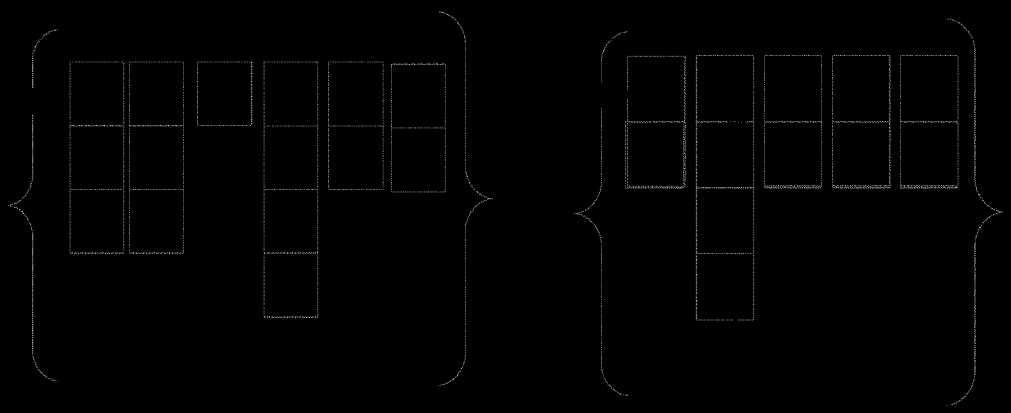
4. (2 puntos) Implementa la función

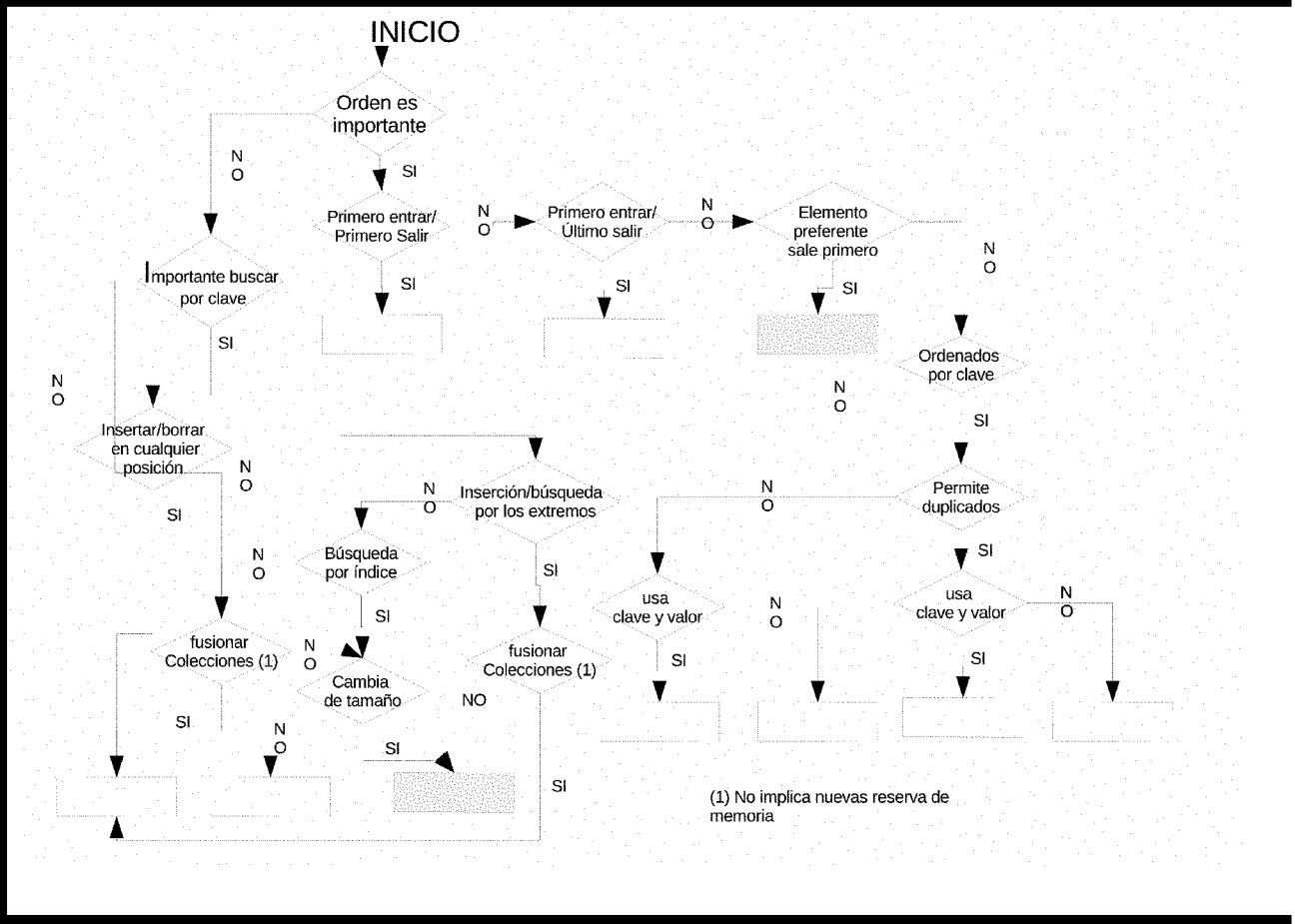
   bintree<T>::node siguiente_nodo_nivel(const bintree<T>::node &n, const bintree<T> &arb)

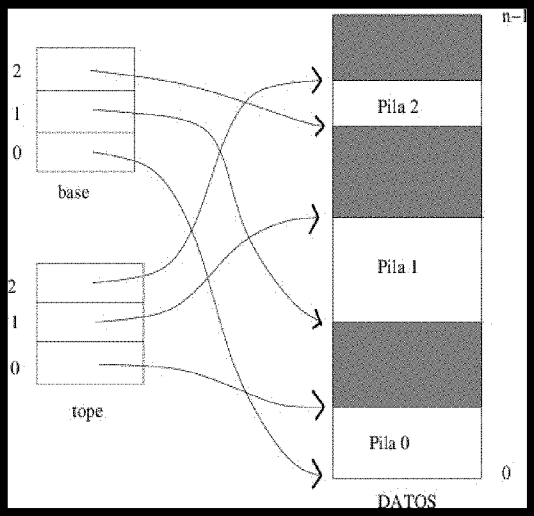
   que dado un nodo n de un árbol arb, devuelve el siguiente nodo que está en ese mismo nivel del árbol.

5. (2 puntos) Un "**q-APO**" es una estructura jerárquica que permite realizar las operaciones *eliminar-minimo* e *insertar* en un tiempo O(log_2(n)), y que tiene como propiedad fundamental que para cualquier nodo X la clave almacenada en X es **menor** que la del hijo izquierda de X y esta a su vez **menor** que la del hijo derecha de X, siendo el árbol binario y estando las hojas empujadas a la izquierda.
   Diseñar una función para insertar un nuevo nodo en la estructura. Aplicarlo a la construcción de un q-APO con las claves {29, 24, 11, 15, 9, 14, 4, 17, 22, 31, 3, 16}.

Tiempo: 3 horas




INICIO
Orden es importante
NO
SI
Primero entrar/ Primero Salir
NO
Primero entrar/ Último salir
NO
Elemento preferente sale primero
NO
SI
SI
SI
Importante buscar por clave
SI
NO
Ordenados por clave
NO
SI
NO
Insertar/borrar en cualquier posición
NO
SI
NO
Inserción/búsqueda por los extremos
NO
Permite duplicados
SI
NO
Búsqueda por índice
SI
SI
usa clave y valor
NO
SI
usa clave y valor
NO
SI
fusionar Colecciones (1)
NO
Cambia de tamaño
SI
fusionar Colecciones (1)
NO
SI
NO
SI
SI
(1) No implica nuevas reserva de memoria


n-1
2
1
0
base
2
1
0
tope
Pila 2
Pila 1
Pila 0
0
DATOS


DECSAI


DECSAI

< >

Ejemplo 1:

Lista inicial: (a,b,c,d)
Lista final: (a,d,b,c,c,b,d,a)

Ejemplo 2:

Lista inicial: (1,2,3,4,5)
Lista final: (1,5,2,4,3,3,4,2,5,1)

$$PE = \sum_{n \in \{hojas\}} 2^{profundidad(n)}$$