

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:

$$V[i] = \begin{cases} M[i] & \text{si } M.find(i) \neq M.end(). \\ v_def & \text{en caso contrario.} \end{cases}$$

- (a) Dar una especificación (sintaxis y semántica) 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** = {1, 3, 4, 7, 7, 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

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 serían:

- `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 encuentran 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 $L2 = [0,1], [12,23]$.

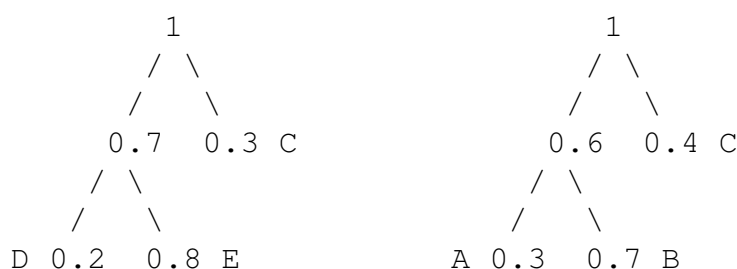
5. (2 puntos) Un árbol de sucesos es un árbol binario donde cada nodo tiene asociada una etiqueta con un valor real en el intervalo $[0,1]$. Cada nodo que no es hoja cumple la propiedad de que la suma de los valores de las etiquetas de sus hijos es 1. Un suceso es una hoja y la probabilidad de que éste 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 raíz 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: $1.0 \cdot 0.6 \cdot 0.3 = 0.18$; B: $1.0 \cdot 0.6 \cdot 0.7 = 0.42$; C: $1.0 \cdot 0.4 = 0.4$). En cambio, en el árbol de la izquierda hay un suceso probable: E, porque $1.0 \cdot 0.7 \cdot 0.8 = 0.56$



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.

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 :

5	4	3
2	1	2
9	0	2
8	9	1

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

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 $m2=\{1,2,3,3,4\}$ entonces $m1 \cap m2 = \{2,3,3\}$ ó si $m1=\{2,2,2,3,3\}$ y $m2=\{1,2,2,2,3,3,4\}$ entonces $m1 \cap m2 = \{2,2,2,3,3\}$
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 en 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

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 **consultar** 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}.

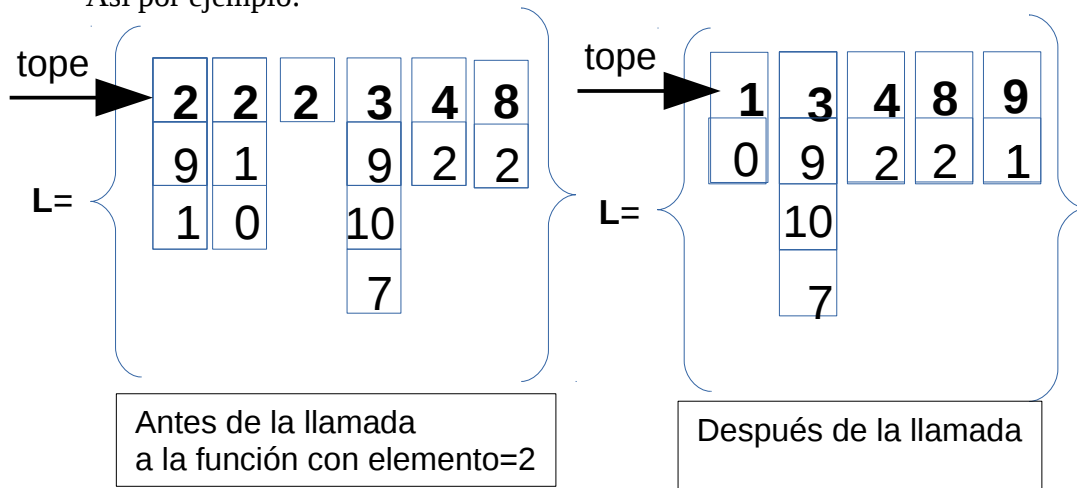
Tiempo: 3 horas

Estructuras de Datos
Curso 2014-2015. Convocatoria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (1 punto) Razonar (en términos de eficiencia de las operaciones) qué TDA elegirías entre los que se indican para:
 - (a) Implementar una **Cola con prioridad** (con funciones básicas de insertar, borrar, y frente) de entre: un **ABB** (árbol binario de búsqueda), un **APO** (árbol parcialmente ordenado implementado mediante un **heap**) y un **Vector ordenado**.
 - (b) Implementar un **Diccionario** (con funciones básicas de insertar, borrar y buscar) de entre: una **lista ordenada**, un **vector ordenado** y un **AVL**.
 - (c) Implementar un **Conjunto no ordenado** (con funciones básicas de insertar, borrar y buscar) de entre: un **AVL**, un **ABB**, y una **Tabla hash abierta**.
2. (1.5 puntos) Dada una lista que contiene pilas con enteros y que se encuentra ordenada de menor a mayor por el tope de cada pila, implementar una función **borrar**, que elimina el tope de cada pila en la lista, con valores iguales al dado como parámetro. La cabecera de la función sería:

void Borrar(list<stack<int> > &L, int elemento)

Así por ejemplo:



Nota: la lista tiene que quedar ordenada por el tope de las pilas tras la ejecución de la función.

3. (1.5 puntos) Implementar una función que muestre las etiquetas de los nodos de un árbol binario de enteros que estén entre dos **niveles** dados **n1** y **n2** ($0 \leq n1 < n2$) ambos inclusive

4. (1 punto) Se dice que un árbol binario de enteros es inferior a otro si (teniendo la misma estructura de ramificación), los elementos del primero, en los nodos coincidentes en posición, son menores que los del segundo. Diseñar una función booleana que dados dos árboles binarios, devuelva true si el primero es inferior al segundo

bool es_inferior(bintree<int> & ab1, bintree<int> & ab2)

- 5.- (1 punto) Implementar un iterador que itere sobre las claves que sean números primos en una clase Diccionario definida como:

```
class Diccionario{
    private:
        map<int, list<string> > datos;
        .....
        .....
}
```

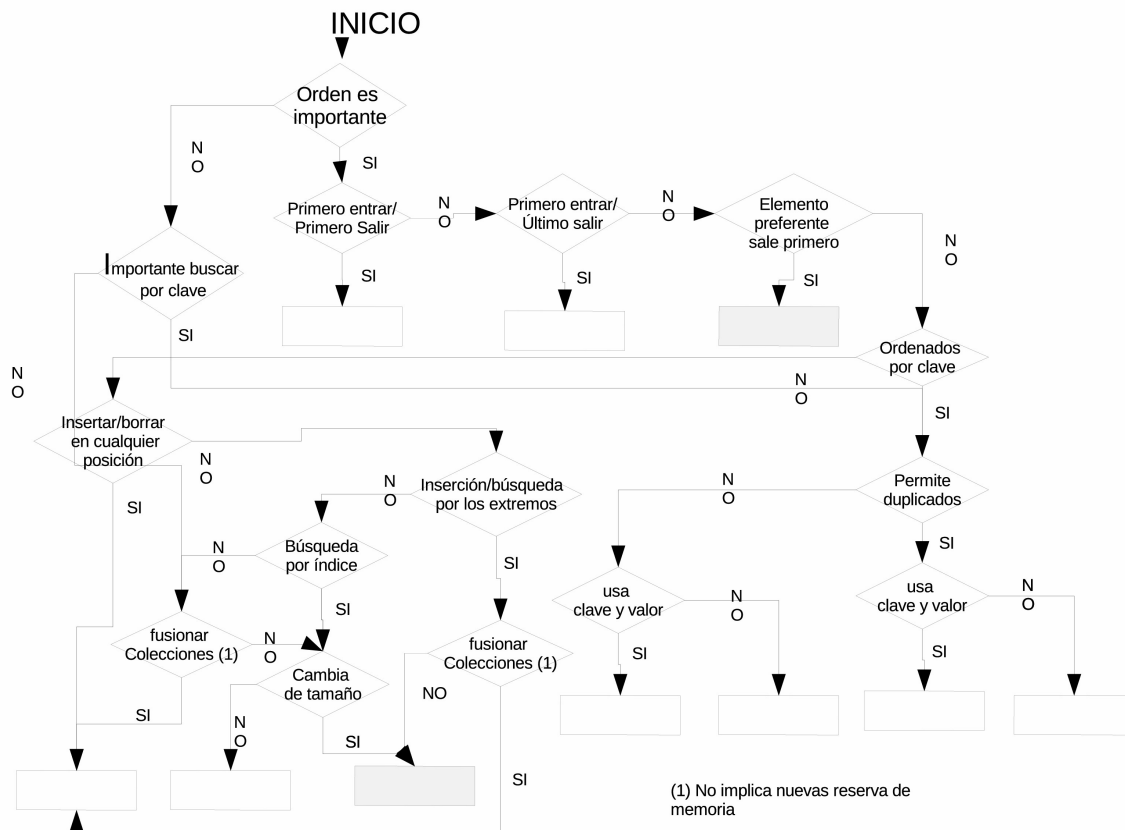
Han de implementarse (aparte de las de la clase iterador) las funciones begin() y end(). Se supone implementada una función **bool primo (int x)** que devuelve true si el entero x es primo.

Tiempo: 3 horas

Estructuras de Datos
Curso 2014-2015. Convocatoria de Septiembre
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

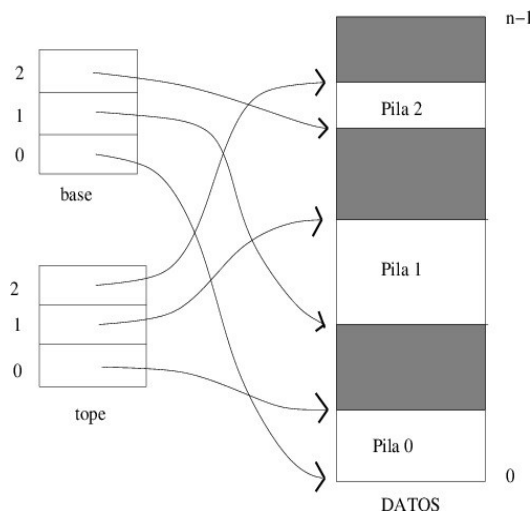
1.-

(a) (1 Punto) Completar la siguiente figura, indicando en los rectángulos la estructura de datos que utilizarías cuando un problema tiene las restricciones indicadas.



(b) (1 Punto) Razona tu decisión sobre los TDA que has ubicado en los recuadros grises en base a la representación interna del mismo así como considerando la eficiencia de las operaciones.

2.- (2 puntos) Se pretende almacenar k pilas en un único vector utilizando la estructura de datos sugerida en la figura siguiente (para el caso $k=3$)
 Implementa las funciones **push** y **pop** que insertan/borran un determinado entero en una de las pilas



3.- (2 puntos) Dada una lista de enteros con elementos repetidos, diseña (usando el TDA **list**) 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. La lista original no se ve modificada. Por ejemplo:

si **entrada**={1,3,4,5,6,3,2,1,4,5,5,1,1,7}, **salida**={{1,1,1},{2},{3,3},{4,4},{5,5,5},{6},{7}}

4.- (2 puntos) Sea T un árbol binario con n nodos cuyas etiquetas (de tipo string) son todas diferentes. Se define un k-nodo como un nodo **v** que cumple la condición de que el número de descendientes en el subárbol izquierdo de **v** difiere del número de descendientes del subárbol derecho al menos en k elementos. Usando el TDA bintree, diseña una función que tenga como entrada un árbol binario de string y como salida el listado de las etiquetas de los 5-nodos que posee

list<string> cinconodos(bintree<string> & ab)

5.- (2 puntos) Implementa un iterador que itere sobre las claves que sean números pares en una clase Diccionario definida como:

```

class Diccionario{
private:
    map<int, list<string> > datos;
    .....
    .....
}
    
```

Han de implementarse (aparte de las de la clase iterador) las funciones begin() y end().

Tiempo: 3 horas

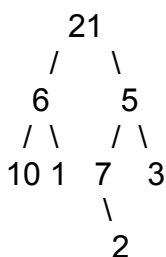
Estructuras de Datos
Curso 2015-2016. Convocatoria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (1 punto) Construye una clase **agenda** que de soporte al manejo de información del tipo persona: <nombre, num_telefono, direccion>. Los nombre son todos distintos y se tiene que poder buscar de forma eficiente tanto por nombre como por número de teléfono o dirección. Además de en tiempo también debe ser eficiente en espacio. Implementa las funciones: persona **buscar_nombre**(string nom), persona **buscar_telefono**(string num), void **insertar**(persona p)
2. (1.5 puntos) Implementa una función bool **sumalqual**(const list<list<int> > &L) que devuelve true si todas las sumas por filas y por columnas da el mismo resultado (se asume que todas las listas tienen igual longitud). Por ejemplo, en la siguiente lista si se suma cada fila y cada columna siempre se obtiene 3

$$L = \begin{matrix} <0,1,1,1>, \\ <2,1,0,0>, \\ <1,0,2,0>, \\ <0,1,0,2> \end{matrix}$$

3. (1.5 puntos) Una **secuencia guiada** sobre un árbol binario de enteros comienza por la raíz para a continuación, en cada paso, seleccionar el nodo con etiqueta más pequeña de entre los nodos disponibles en ese momento que no hayan sido listados. Se entiende por **nodo disponible** aquel cuyo padre ya ha sido procesado (excluyendo la raíz).
 - a) ¿Qué estructura auxiliar utilizarías para almacenar el conjunto de nodos disponibles? Razona tu respuesta.
 - b) Implementar una función que permita obtener una secuencia **guiada** de un árbol binario.

Ejemplo:



Recorrido guiado: 21, 5, 3, 6, 1, 7, 2, 10

4. (1 punto) Tenemos un contenedor de pares de elementos, {clave, vector<int>} definida como:

```
template <typename T>
class contenedor{
private:
    unordered_map<T, vector<int> > datos;
    .....
    .....
}
```

Implementa un iterador que itere sobre los elementos que cumplan la propiedad de que la suma de los elementos del vector<int> sea un número impar. Han de implementarse (aparte de las de la clase iteradora) las funciones begin() y end().

5. (1 punto) Implementa una función que compruebe si un árbol binario es simétrico de otro.

bool simetricos(const bintree<T> &a, const bintree<T> &b).

Ejemplo:



Tiempo: 3 horas

Estructuras de Datos
Curso 2015-2016. Convocatoria de Septiembre
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (2 puntos) Un almacén está desarrollando una aplicación que permita un acceso rápido a los productos de los que disponen y no se tiene claro qué estructura de datos elegir para almacenar los registros (nombre, código, año...). Las operaciones que son críticas para el almacén son: a) Inserción; b) búsqueda, que se hace por el nombre del producto; c) borrado, a partir del código único asociado a cada producto; d) imprimir de forma ordenada todos los productos. Analizar la eficiencia de las 4 operaciones si se usa como estructura de datos: Un vector ordenado, una lista ordenada, un ABB, un AVL y una Tabla hash.

¿Qué representación utilizarías para garantizar la mejor solución al problema?.

2. (2 puntos) Se dispone de un conjunto de claves de gran tamaño, que se almacenan de forma ordenada. La particularidad de este conjunto es que las claves pueden aparecer de forma consecutiva un número indefinido de veces. Por ej:

$\{ 1,1,1,1,1,3,3,3,6,6,6,6,6,6,9,9,9,9,9,..... \}$

a) Para el acceso a las claves se pretende diseñar un TDA con la misma especificación del vector de la STL, sin embargo se busca una representación que almacene los datos con un espacio de memoria proporcional al número de claves distintas existentes. ¿Qué representación propondrías?

b) Considerando dicha representación, construye una función que dada una posición i del vector original devuelva el valor almacenado en esa posición.

3. (2 puntos) Usando la clase `list<T>`, construye la siguiente función que permite "duplicar" una lista (intercalando alternativamente tras cada elemento en la posición i , el elemento que está en la posición $n-i-1$ ($i=0,1,...,n-1$)).

```
template <typename T>
void duplicar(const list<T> & inicial, list<T> & final)
```

Ejemplo 1:

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

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

Ejemplo 2:

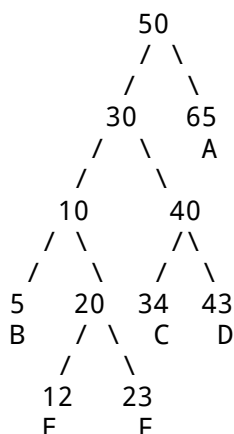
Lista inicial: (1,2,3,4,5)

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

4. (2 puntos) Usando el TDA bintree, diseña una función para calcular la potencia externa PE de un ABB definida como:

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

P.ej. El árbol mostrado a continuación tiene una potencia externa de 58.



$$PE = A+B+C+D+E+F = 2^1 + 2^3 + 2^3 + 2^3 + 2^4 + 2^4 = 58$$

5.- (2 puntos) Implementa un iterador que itere sobre las claves que sean números impares en una clase diccionario definida como:

```
class diccionario{
private:
    map<int, list<string> > datos;
    .....
    .....
}
```

Además de los métodos de la clase iterador, debes implementar begin y end de la clase diccionario.

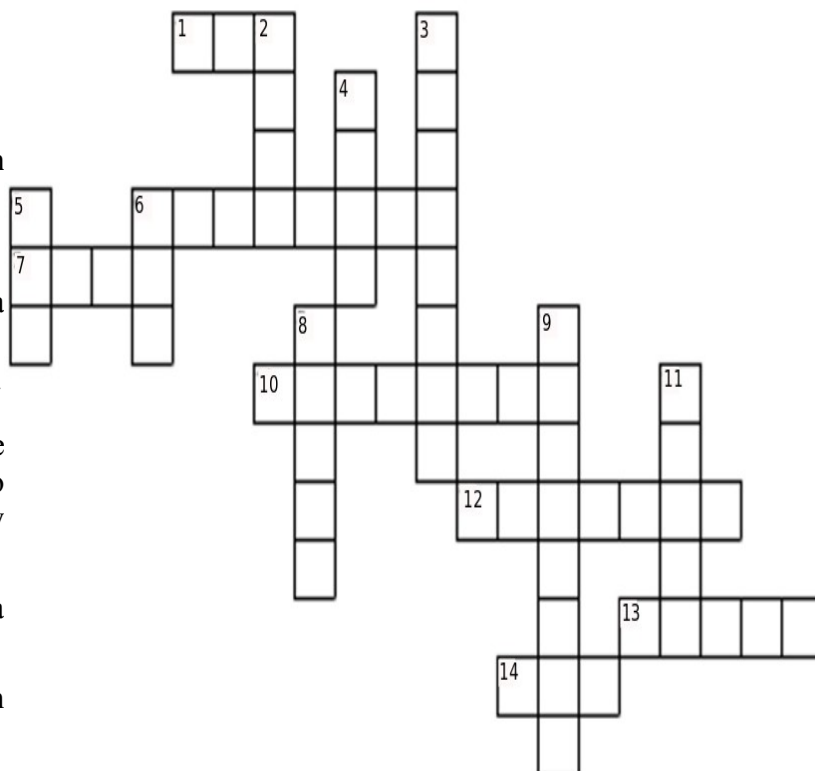
Tiempo: 3 horas

Estructuras de Datos
Curso 2016-2017. Convocatoria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. Rellenar el siguiente crucigrama
 (0.5 puntos)

Horizontal:

1. Árbol binario equilibrado
6. TDA (STL) que representa un conjunto de claves ordenadas que admite claves repetidas
7. Estructura dónde se consulta y borra el último elemento insertado
10. T.D.A para recorrer un contenedor
12. Recorrido en un árbol binario que lista de forma recursiva primero el hijo a la izquierda, luego la raíz y finalmente el hijo derecha
13. TDA (STL) muy eficiente para insertar/borrar por los extremos
14. TDA (STL) que representa un conjunto de claves ordenadas que no admite claves repetidas



Vertical

2. TDA (STL) que admite inserciones en cualquier posición
3. TDA (STL) que contiene pares <clave, información asociada>, ordenados por clave y que admite claves repetidas.
4. Tablas para alcanzar tiempos de búsqueda constantes.
5. Árbol binario completo en el que la etiqueta de un nodo tiene valores más pequeños que las de sus hijos.
6. TDA (STL) que contiene pares <clave, información asociada>, ordenados por clave y que no admite claves repetidas.
8. TDA (STL) en el que se consulta/borra por el tope.
9. Recorrido en un árbol binario en el que se lista de forma recursiva en primer lugar la raíz y a continuación su hijo izquierda y su hijo derecha
11. T.D.A (STL) en la que la consulta y borrados se realizan por un extremo y la inserción por el otro.

2. (1.5 puntos) Se define un árbol **RV** a un árbol **binario** en el que cada nodo tiene una etiqueta que puede tomar los valores Rojo (R) o Verde (V). Un nodo V indica que se puede pasar por dicho nodo, mientras que un nodo R bloquea el paso. Se pide diseñar un método que devuelva una lista con aquellos nodos verdes, v, del árbol que cumplan la propiedad de que: o v es hoja o sus dos descendientes son rojos (R)

3. (a) (0.5 puntos) Insertar las siguientes claves en un **AVL**:

44, 29, 47, 64, 48, 50, 80, 36, 05, 61, 51, 72

Borrar el elemento 48 del árbol

(b) (0.5 puntos) Insertar los elementos anteriores en una **tabla hash cerrada** con resolución de colisiones lineal ¿Qué valor del tamaño de la tabla considerarías?

4. (1 punto) Implementar un método que chequee si un **bintree** que representa un **árbol binario de búsqueda (ABB)** es correcto

5. (1 punto) Un tipo de dato **pila_doble** es un tipo que permite insertar y borrar por ambos extremos. Implementar las operaciones push y pop de este tipo de dato usando como representación un vector. Las operaciones tienen un parámetro que indica la pila sobre la que se hacen las operaciones:

a) **void pila_doble<T>::push(int numpila, const T & elem)**

b) **void pila_doble<T>::pop(int numpila)**

6. (1 punto) Se define una **matriz bilineal simétrica** como una **matriz nxn** en la que todos los elementos significativos (distintos de un valor por defecto) están situados en las 2 diagonales principales y tal que al recorrer ambas diagonales en orden creciente de filas, presentan los mismos elementos. Un ejemplo de este tipo de matrices es la matriz del ejemplo. Es una matriz **6x6** de valores enteros y con el **0** como valor por defecto.

2	0	0	0	0	2
0	4	0	0	4	0
0	0	7	7	0	0
0	0	1	1	0	0
0	9	0	0	9	0
6	0	0	0	0	6

Proponer una representación eficiente para la matriz e implementar:

- a) el método **int at(int fil, int col)** que devuelva el elemento en la fila fil, columna col.
 b) un iterador que itere sobre los elementos significativos. Irían primero todos los elementos significativos de la primera fila, a continuación los elementos significativos de la segunda fila y así sucesivamente. Han de implementarse (aparte de las de la clase iteradora) las funciones **begin()** y **end()**.

Duración 3 horas

Estructuras de Datos
Curso 2016-2017. Convocatoria de Julio
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (1 punto) Razonar **detalladamente** (en términos de eficiencia de las operaciones) qué TDA elegirías entre los que se indican para:

- (a) Implementar un **conjunto ordenado de enteros** (con funciones básicas de **insertar, borrar, y buscar**) de entre: un **ABB** (árbol binario de búsqueda no balanceado), un **APO** (árbol parcialmente ordenado implementado mediante un **heap**) y un **vector ordenado**.
- (b) Implementar un **conjunto no ordenado de enteros** (con funciones básicas de **insertar, borrar y buscar**) de entre: un **AVL**, un **vector ordenado**, y un **APO** (árbol parcialmente ordenado implementado mediante un **heap**).

2. (2.5 puntos) Una lista **L** se dice que está ordenada de forma ascendente por índices (orden indexado) si la secuencia $\{L(\text{indices}(0)), L(\text{indices}(1)), \dots, L(\text{indices}(\text{indices.size}()-1))\}$. (siendo $L(i)$ el elemento en la posición i -ésima), es una secuencia ordenada de forma ascendente. Necesariamente **indices** debe ser una lista de enteros que indica una secuencia de índices en **L**. Implementa una función booleana que reciba **L** (**L** puede contener cualquier tipo de dato) y un índice **I** y devuelva **true** si **L** está ordenada de forma ascendente con respecto al índice. Ejemplo: Para $L = \{2.2, 7.1, 3.4, 9.0, 6.2, 5.1\}$ e $I = \{0, 2, 5, 4, 1, 3\}$, la función devolvería **true**.

3. (2.5 puntos) Implementa una función booleana que devuelva **true** si un árbol binario de enteros **ab2** coincide con un subárbol de otro **ab1** (deben coincidir también los valores de las etiquetas)

bool es_subarbol(const bintree<int> & ab1, const bintree<int> & ab2)

4.- (1.5 punto) Implementa un **TDA cola** usando **dos pilas**.

5.- (2.5 puntos) Se dispone de un TDA **indice** donde para cada palabra se almacenan los números de página en los que aparece en un libro. Se desea recorrer el índice considerando únicamente las ocurrencias en páginas impares de las palabras del contenedor. Para implementar el índice se ha seleccionado la siguiente estructura:

```
class indice{
private:
    map<string, set<int> > datos;
    .....
    .....
}
```

Se pide implementar el TDA **indice::iterator** deseado, de forma que el **operator*** devuelva tanto la palabra como el número de página (que ha de ser necesariamente impar).

Además de los métodos de la clase iterador, se deben implementar las funciones **begin()** y **end()**.

Tiempo: 3 horas