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** = $\{1, 3, 4, 7, 7, 9, 11, 16, 25, 19, 14, 8, 9\}$.

   (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