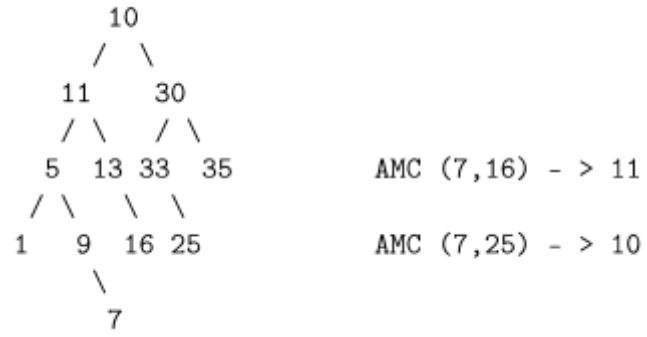
Estructuras de Datos. Grupo D.
Relación de Problemas. Árboles.

1. 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 sí mismo). Diseñar una función que tenga como entrada un árbol binario de enteros y dos nodos v y w como salida y el AMC de v y w.
   Ejemplo:



2. 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 sólo hijo, éste ha de situarse necesariamente a la izquierda).
   Se pide:
   1. Implementar un método que compruebe si un árbol binario A está sesgado hacia la izquierda.
   2. Implementar un método que transforme un árbol binario en un árbol 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 árbol transformado, por lo que no es válido hacer un intercambio de las etiquetas de los nodos

3. 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:
   1. Diseñar una función que compruebe si un árbol binario A es un árbol de sucesos. Su prototipo será

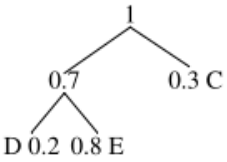
```
bool check_rep (const bintree<float> &A)
```

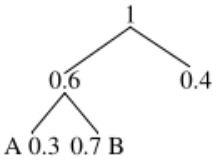
   2. Diseñar una función que indique si existe algún suceso probable en el árbol de probabilidades A. Su prototipo será:

```
bool probable (const bintree<float> &A)
```

   Por ejemplo: en el árbol de probabilidades de la derecha no existe un suceso probable porque los tres sucesos tienen probabilidad inferior a 0.5:
   1. A: 1;0 _ 0;6 _ 0;3 = 0;18
   2. B: 1;0 _ 0;6 _ 0;7 = 0;42
   3. 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





4. 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);
```

5. Dado un bintree<int> T, implementar una función

```
void prom_nivel(bintree<int> &T, list<float> &P);
```

que genere una lista de reales P, donde el primer elemento de la lista sea el promedio de los nodos del árbol de nivel 0, el segundo sea el promedio de los de nivel 1, el tercero el promedio de los de nivel 2, y así sucesivamente. Es decir, que si el árbol tiene profundidad N, la lista tendrá N+1 elementos de tipo float.

6. Implementar una función

```
bool desordenequal (const bintree<int> & A, const bintree<int> & B);
```

que reciba dos árboles binarios y devuelva true si son “desordenadmente” iguales. Se dice que dos árboles son “desordenadamente” iguales si:
(a) las raíces de ambos son iguales,
(b) el conjunto de hijos de la raíz de uno es igual al conjunto de hijos de la raíz del otro (Notar
que en un “conjunto” no importa el orden, por lo que esta condición implica que las raíces tienen los mismo hijos, aunque podrían estar en diferente orden.), y
(c) la condición se cumple recursivamente para cada par de subarboles de cada par de nodos
equivalentes (uno de cada bintree).
Como estructura auxiliar solo se permite usar un <map>
Algoritmo sugerido: sean A y B dos árboles binarios de enteros (bintree<int>), dados dos iteradores itA e itB apuntando a las raíces de A y B respectivamente: Colocar todos los hijos del nodo itA en un map MA, donde las claves sean la etiquetas de dichos hijos, y los valores sean iteradores apuntando a los mismos. Colocar todos los hijos del nodo itB en un map MB, donde las claves sean las etiquetas de dichos hijos, y los valores sean iteradores apuntando a los mismos. Si los map MA y MB tienen diferente tamaño retornar false. Recorrer en simultáneo MA y MB, y por cada par de pares clave-valor (uno de MA y otro de MB):

- Si las etiquetas (claves) son diferentes retornar false
- Aplicar el algoritmo recursivamente utilizando los dos iteradores (valores)

7. 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. Implementar una función booleana que dados dos árboles binarios, devuelva true si el
primero es inferior al segundo:
bool es_inferior(const bintree<int> &ab1, const bintree<int> &ab2);

8. Implementar una función:
bool es menor(bintree<int>&A,bintree<int>&B);
que devuelve true si A < B, con A y B árboles binarios no vacios. Se entiende que A<B si:
(a<b) ó
(a=b) && (Ai<Bi) ó
(a=b) && (Ai=Bi) && (Ad<Bd)

9. Implementar una función:
int nodos_k (bintree<int> &A, int k);
que devuelve el número de nodos de un árbol binario cuya etiqueta es exactamente igual a k.

10. Implementar 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 binario arb, devuelve el siguiente nodo que está en ese mismo nivel del árbol. Si es el último nodo del nivel devuelve el primero del siguiente nivel
11. Construir un AVL 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.
12. 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}
13. Un "q-APO.es una estructura jerárquica que permite realizar las operaciones eliminar-mínimo 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. Implementar una función para insertar un nuevo nodo en la estructura y aplicarla a la construcción de un q-APO con las claves {29, 24, 11, 15, 9, 14, 4, 17, 22, 31, 3, 16}.

14. Se denomina “odanedro” T* de un ABB T, al arbol binario construido de forma que todo hijo izquierda de un nodo en T pasa a ser hijo derecha de T* y todo hijo derecha de un nodo en T pasa a ser hijo izquierda de ese nodo en T* . Implementar una función que tenga como entrada un ABB y devuelva su“odanedro”. ¿Cómo habría que recorrer el “odanedro” para que las etiquetas de los nodos se listen en orden ascendente de valor?