UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Reto 5: TDA No Lineales II

J. Fdez-Valdivia
Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

Resolver 5 de los siguientes ejercicios:

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

void prom_nivel(tree<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.

2. Implementar una función

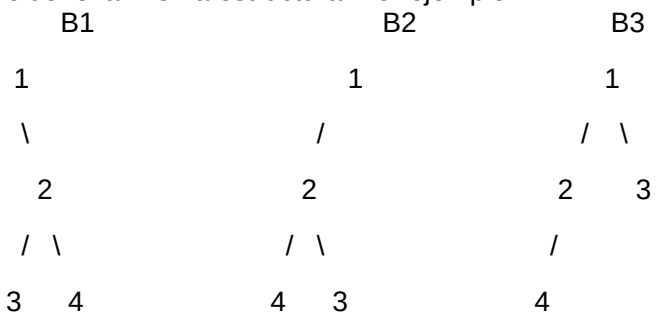
bool desordenequal (const bintree<int> A, const bintree<int> B);

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 la 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),

3. Se dice que un arbol 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(bintree<int> & ab1, bintree<int> & ab2);

4. Dos árboles binarios B1, B2 son isomorfos si se pueden aplicar una serie de inversiones entre los hijos derechos e izquierdos de los nodos de B2 de manera que quede un árbol semejante a B1, es decir que tiene la misma estructura. Por ejemplo



B1 es isomorfo a B2 pero no a B3. En particular si dos árboles son isomorfos la cantidad de nodos que tienen una profundidad y altura dadas es la misma. Por ejemplo en el caso anterior B1 y B2 tienen 4 nodos, de los cuales uno a profundidad 1, 2 a profundidad 2 y por supuesto 1 a profundidad 0 (raíz).

Concretamente B1 es isomorfo a B2 si: (a) Ambos son vacios

(b) Los subárboles de los hijos izquierdos de B1 y B2 son isomorfos entre sí y los derechos también ó el subárbol del hijo izquierdo de B1 es isomorfo al derecho de B2 y el derecho de B1 es isomorfo al izquierdo de B2.

Implementar una función

bool btisomorph(bintree<int> &B1,bintree<int> &B2);

que devuelva true si B1 y B2 son isomorfos y false en caso contrario

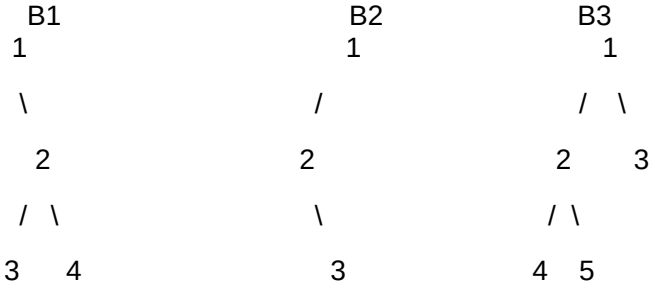
Nota: Notar que sólo se mira la estructura del árbol, no los valores de los nodos.

5. Implementar una función

int onechild(bintree<int> &T);

que dado un arbol binario T, devuelva cuantos nodos de T tienen exactamente un solo hijo

Ejemplos:



onechild(B1) devuelve 1, onechild(B2) devuelve 2, onechild(B3) devuelve 0

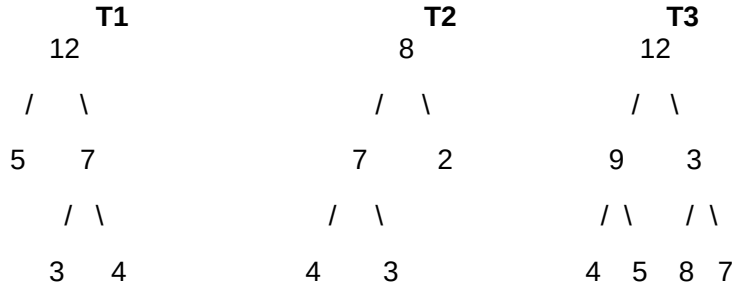
6. Implementar una función

bool is_recurrent_tree(bintree< int > &T);

que devuelva true si T es un árbol recurrente.

Un árbol binario se considera recurrente si es homogéneo y cada uno de sus nodos interiores es la suma de sus 2 nodos hijos.

Por ejemplo



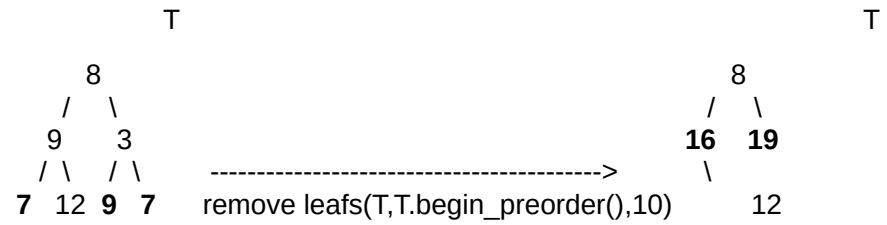
T1 -> true

T2 -> false (la raíz no es la suma de sus hijos)

T3 -> false (el nodo con valor 3 no es la suma de sus hijos

7. Implementar una función:
   int remove leafs(bintree<int>&T, bintree<int>::preorder_iterator n, int min_leaf_val)

   que transforma un subárbol de un árbol binario, eliminando las hojas cuyo valor es menor que un cierto umbral min_leaf_val . A medida que las hojas son eliminadas, su valor debe ser acumulado en el padre, de manera que la suma de los valores en los nodos del subárbol se mantiene. Por ejemplo:



   Puede pasar que si la suma de los valores del subárbol es menor que min_leaf_val entonces el subárbol completo es eliminado, en ese caso el valor de retorno es la suma de los valores del subárbol eliminado. Si el árbol no es eliminado entonces el valor de retorno es nulo. En resumen Si n = nodonulo, remove_leafs retorna 0. En caso contrario, se aplica remove_leafs a los hijos izquierdo y derecho de n. (Tener en cuenta que estas llamadas pueden eliminar el nodo correspondiente, por lo tanto debe prestarse atención a los iterators inválidos). El valor retornado por los hijos debe acumularse en n.

   Después de la aplicación recursiva de remove_leafs a los hijos de n:

   • Si n no es hoja, devolverá 0.

   • Si el valor de n no es menor que el umbral min_leaf_val, devolverá 0.

   • Finalmente, si lo anterior no se cumple, se debe eliminar n y devolver su valor.

8. Dibujar el árbol binario cuyos recorridos **preorden y postorden** son:

   (a) PRE={Z,A,R,Q,L,M,N,T,S,W,Q} POST ={Q,L,R,N,M,A,W,Q,S,T,Z}.

   (b) PRE={A,B,C,E,F,Z,Y,W,G,D} POST ={B,E,Z,Y,W, F,G,C,D,A}.

   (c) PRE={B,F,E,H,I,D,J, A,C,G}, POST={E, F, I,D,A,J,C,H,G,B}.

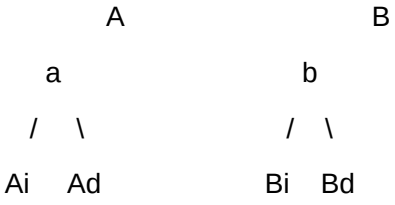
   (d) PRE={C,Z,Q,R,A,M, P,K,L,T}, POST ={Z,A,P,M,K,L,R,T,Q,C}.

   (e) PRE={W,A,B,D,E,H, J,C,F,G}, POST ={D,J,H,E,B,F,G,C,A,W}

Dar el recorrido en **inorden** en cada caso.

9. 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**

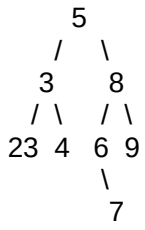
(a=b) && (Ai=Bi) && (Ad<Bd)

10. Implementar una función:

bintree<double>::preorder_iterator encuentra_suma(double s, bintree<double> &A);

que devuelve el iterador al nodo m del árbol binario A tal que la suma de las etiquetas de todos los nodos descendientes de m (incluyendo a m) es s. Si no existe tal nodo, entonces debe retornar null(). Si hay varios nodos que cumplen la condición basta con que devuelva uno de ellos.

Por ejemplo: **A**



encuentra_suma(65,A) devuelve el iterador al nodo raiz

encuentra_suma(30,A) puede devolver el iterador al nodo con etiqueta 8, ó etiqueta 3

encuentra_suma(9,A) devuelve un iterador al nodo con etiqueta 9

encuentra_suma(10,A) devuelve un iterador null()

Restricciones: El algoritmo debe ser O(n) donde n es el número de nodos en el árbol.

11. 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.

12. Implementar la función:

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

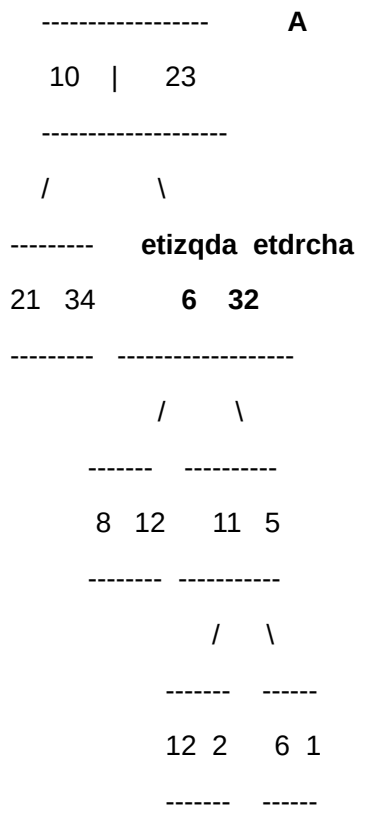
que dado un nodo n de un árbol binario arb, devuelve el siguiente nodo que está en ese mismo nivel del árbol.

13. Implementar una función:

void bi_listado (bintree<int>::node n, bintree<int> A);

que imprima el “bi-listado” de un árbol binario que se define como una combinación de los listados preorden y postorden. Asumamos que las etiquetas tienen dos partes, una etiqueta “derecha” y una “izquierda”, entonces el bi-listado de un nodo n se define como la lista vacía si

el nodo es nulo y, si no, recursivamente como la etiqueta izquierda de n, seguida del bi-listado de los hijos, seguido de la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el bi-listado es la siguiente lista: {10, 21, 34, 6, 8, 12, 11, 12, 2, 6, 1, 5, 32, 23}.

Notar que si consideramos sólo las etiquetas izuierdas, entonces el resultado es el preorden, mientras que si consideramos sólo las derechas obtenemos el postorden.

14. Dado un árbol binario de enteros (positivos y negativos) implementar una función

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

que obtenga el número de caminos, en los que la suma de las etiquetas de los nodos que los componen sumen exactamente k.

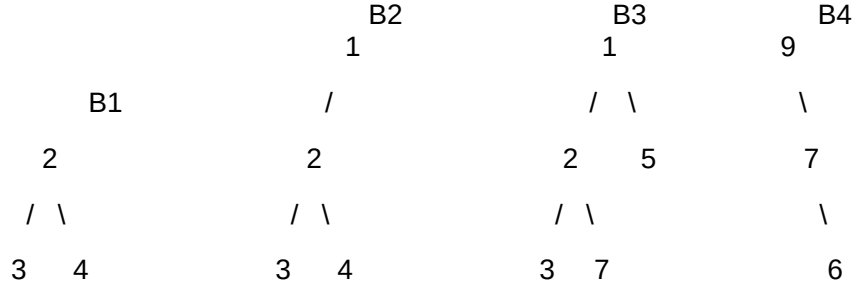
15. Implementar una función

bool incluido(bintree<int>&A,bintree<int>&B);

que devuelve true si la estructura jerárquica del árbol del nodo B está “incluida” dentro del ´arbol A,

independientemente de las etiquetas de los nodos correspondientes.

Ejemplo:



B1 está incluido en B2 y en B3, B2 está incluido en B3, B1 no está incluido en B4, B4 no está incluido en B2.

16. Implementar la función:

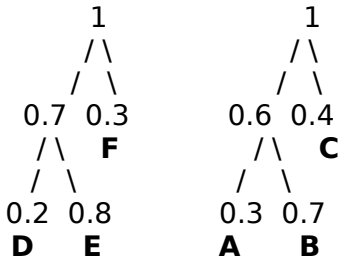
void Fibonacci Trees(vector<bintree<int>> v);

que construye la sucesión de árboles binarios de Fibonacci y los almacena en un vector de árboles. La sucesión comienza con un árbol con 1 solo nodo (To) y un árbol con solo un hijo a la izquierda (T1). A partir de ellos, se construye la sucesión construyendo cada árbol binarto Ti insertando Ti-1 a la derecha y Ti-2 a la izquierda (i=2,..,n)

Ejemplo:

17. Un árbol de probabilidades 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) Implementar una función que compruebe si un árbol binario A es un árbol de probabilidades: **bool es-un-arbol-prob (const bintree<float> & A);**
(b) Implementar una función que indique si existe algún suceso probable en el árbol de probabilidades A.: **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



18. Sea T un árbol binario de enteros con n nodos. 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 en al menos k. Usando el TDA ArbolBinario, implementar una función

list<int> knodos (bintree<int> & A, int k);

que tenga como entrada un árbol binario de enteros y como salida el listado de las etiquetas de los 5-nodos que posee.

19. Implementar una función:

int infancestral (list<int> & preorden, list<int> & postorden, bintree<int> & A, bintree<int>:: node n1, bintree<int>:: node n2);

que dadas dos listas de nodos correspondientes al recorrido en preorden y postorden de un árbol binario y dos nodos n1,n2 pertenecientes a dicho árbol, devuelva 1 si n1 es descendiente de n2, 0 si n2 es descendiente de n1 y -1 en otro caso

20. Implementar una función,

int max_subtree(bintree<int>&T);

que devuelva la suma de etiquetas máxima de entre todos los posibles subárboles del árbol binario T.

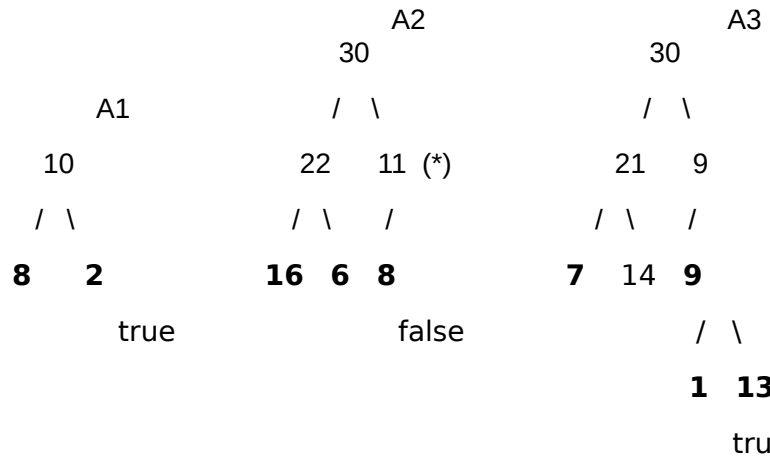
Nota: Las etiquetas de los nodos pueden ser negativas y positivas (el subárbol vacío puede ser el de mayor suma, que se considera 0).

21. Implementar una función:

bool pesointerior (bintree<int> a);

que devuelva true si la etiqueta de los nodos interiores de a es la suma d ellas etiquetas de las hojas que cuelgan de ellos

Ejemplo:



22. Se dice que un arbol 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)

23. Implementar una función que devuelva en un vector las etiquetas de los nodos de un arbol binario de enteros que estén entre dos **niveles** dados **n1** y **n2** (0<=n1<n2) ambos inclusive

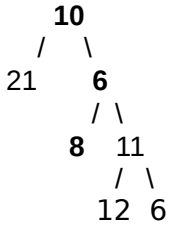
vector <int> labelinterlevel (bintree<int> & A, int n1, int n2);

24. Implementar una función

int sumaparantec (bintree<int> & a, bintree<int>::node n);

que devuelva la suma de las etiquetas de los nodos tales que su etiqueta y la etiqueta de todos los antecesores es par. Si la etiqueta de un nodo es impar, devuelve 0.

Ejemplo:



Devuelve 24

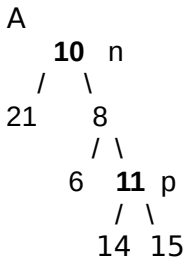
Las etiquetas 6 y 12 no son incluidas ya que su antecesor 11 es impar

25. Implementar una función

bintree<int>:: node cont_hijos (list <int> & L, bintree<int> & A);

que devuelva un nodo en el árbol a cuyas etiquetas de hijos (de izquierda a derecha) coincidan con los enteros de la lista L. Si no hay ningún nodo así, devuelve el nodo nulo.

Ejemplo:



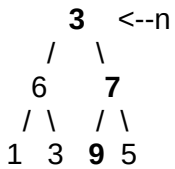
L={21,8} ----------> nodo n
L={14,15} ----------> nodo p
L={11,6} ----------> nodo nulo
L={6,9} ----------> nodo nulo

26. Dado un árbol binario A y un nodo n en el mismo, implementar una función:

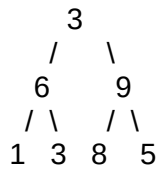
bool camino_ord (bintree<int> A, bintree<int> :: node n);

que devuelva true si existe algún camino desde n a una hoja con los elementos ordenados.

Ejemplo:



true



false

Idea: Para un nodo dado n debe devolver true si el nodo es una hoja o bien alguno de sus hijos p contiene un camino ordenado y ***p > *n**

27. Implementar una función:

bool depth_if(bintree<int> & A);

que devuelva true si el nodo a mayor profundidad en A tiene etiqueta par y false en caso contrario

Ejemplo:

true

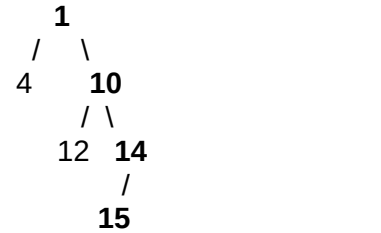
false

28. Implementar una función

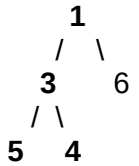
void longest_path(btree<int> &A,list<int> &L);

que, dado un árbol binario A, devuelva en L la lista de valores en el camino más largo en el árbol. Si hay más de un camino, devuelve cualquiera de ellos.

Ejemplo:



Devuelve: L= {1,10, 14, 15}



Devuelve: L={1,3,5} ó L={1,3,4}

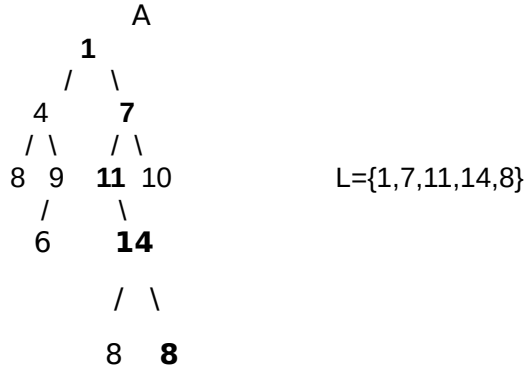
29. Implementar una **función no recursiva** para recorrer un árbol binario en **inorden**

30. Implementar una función

void path_of_largest(tree<int> &A,list<int> &L);

que, dado un árbol binario A , devuelva en L el camino que se obtiene recorriendo el árbol desde la raíz a las hojas siempre por el hijo con mayor valor de la etiqueta. Si para un nodo, el valor más grande está repetido, entonces el camino puede seguir por cualquiera de los hijos.

Ejemplo:



L={1,7,11,14,8}

31. 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

32. Implementa una función booleana que devuelva true si la estructura de ramificación de un árbol binario de enteros ab2 coincide con la de algún subárbol de otro árbol binario ab1 (deben coincidir también los valores de las etiquetas)

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

33. Implementar una función para determinar si un árbol binario tiene **más de un camino** desde una hoja a la raíz cuya suma de etiquetas sea igual a k.

bool suma_k(const bintree<int> &arb, int k)

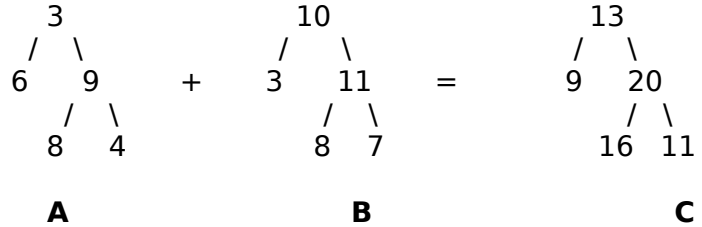
34. Sea una operación f que opera sobre un árbol binario A y que para cada recorrido r € {Preorden, Inorden, postorden} procesa los nodos en orden inverso al que indica r y sea g una operación que procesa nos nodos en el mismo orden que indica el recorrido r € {Preorden, Inorden, postorden} sobre el árbol reflejado de A (árbol en el que el hijo a la izquierda de un nodo del original pasa a ser el hijo a la derecha del nodo en el reflejado y viceversa). ¿Para que recorrido(s) es cierto que f(r)=g(r)?

35. Implementar una función:

void sumadosarboles (bintree<int> A, bintree<int> B, bintree<int> C);

que dados 2 árboles binarios A y B con la misma estructura de ramificación, devuelva un árbol binario C, con la misma estructura y con etiquetas suma de las etiquetas de A y B en los nodos correspondientes.

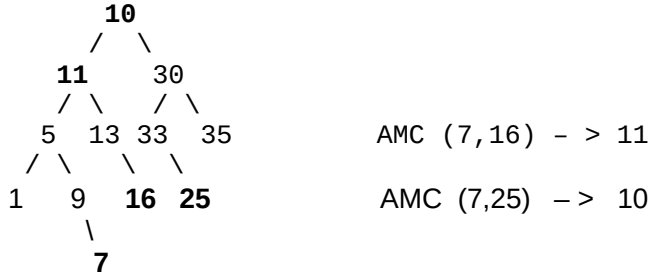
Ejemplo:



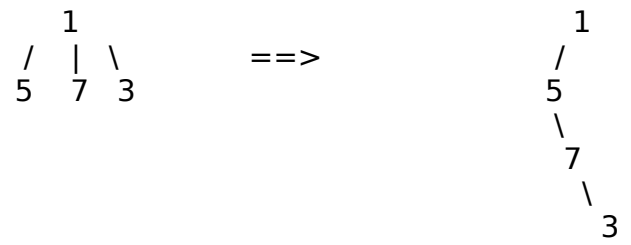
36. Implementar una función:

bintree<int>:: node AMC (bintree<int> &A, bintree<int>:: node v, bintree<int>:: node w);

que dados 2 nodos v y w de un árbol binario A, devuelva su ancestro común más cercano (nodo de mayor profundidad que tiene tanto a v como a w como descendientes, entendiendo como caso límite que un nodo es descendiente de sí mismo.



37. Podemos “convertir” un árbol general en un árbol binario, haciendo que tenga la misma raíz, que el hijo más a la izquierda de cada nodo en el árbol general (cuando exista) se convierta en el hijo a la izquierda del nodo en el árbol binario y que el hermano a la derecha de cada nodo en el árbol general (cuando exista) se convierta en el hijo derecha del nodo en el árbol binario.

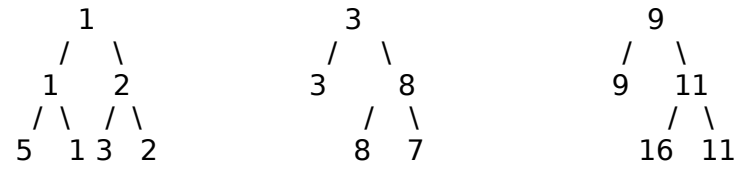


¿Cual es la relación entre los recorridos del árbol general y del binario asociado? Razona la respuesta.

38. Un árbol de selección es un árbol binario en el que cada nodo tiene la etiqueta del menor de sus 2 hijos. Implementar una función:

bool selección (bintree<int> & A);

que devuelva true si A es árbol de selección y false en caso contrario
Ejemplos:



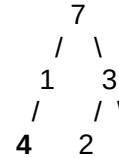
True False True

39. Se defina la “trayectoria” de una hoja en un árbol binario como la suma de las etiquetas en el camino desde la raiz a dicha hoja multiplicadas por el nivln en que está cada nodo en ese camino. Implementar una función:

bintree<int>:: node trayectoria (bintree<int> &A);

que devuelva la hoja con mayor valor de trayectoria

Ejemplo:



Tray(4) = 4*3+1*2+7*1=**21**
Tray(2) = 2*3+3*2+7*1=19
Tray(1) = 1*3+3*2+7*1=16

la hoja con mayor trayectoria es la hoja con etiqueta 4

40. Implementar una **función no recursiva** para recorrer un árbol binario en **preorden**

41. Implementar una función

int enquenivel (bintree<int>:: node w, bintree<int> & A);

que devuelva el nivel del nodo w en el árbol A (la raíz está a nivel 0).

42. Se define la frontera de un árbol binario como la sucesión de sus hojas escrita en orden de izquierda a derecha. Dados 2 árboles binarios de enteros A y B, Implementar una función:

bool tienenigualfrontera(bintree<int> A, bintree<int> B);

que devuelva true si A y B tienen la misma frontera.

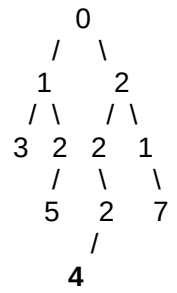
43. Dado un árbol binario, en cuya raíz se encuentra situado un tesoro y cuyos nodos internos pueden contener un dragón o no contener nada, implementar una función:

bintree<int>:: node dragones (bintree<int> & A);

que devuelva la hoja del arbol cuyo camino hasta la raíz tenga el menor numero de dragones. En caso de que existan varios caminos con el mismo numero de dragones, el algoritmo devolverá el que se encuentre más a la izquierda de todos ellos. El árbol tiene como mínimo un nodo raíz y un nodo hoja diferente de la raíz.

La raíz contiene el entero 0, que representa al tesoro.
Los nodos internos contienen el entero 1 para indicar que en el nodo hay un dragón o el entero 2 para indicar que no hay dragón.
En cada hoja se almacena un entero mayor o igual a 3 que no puede estar repetido.

Por ejemplo, dado el siguiente árbol el algoritmo devolverá el entero 4.



44. Se define el volumen de un árbol binario como el número de hojas multiplicado por la altura del árbol. Implementar una función:

bool tienenigualvolumen(bintree<int> A, bintree<int> B);

que devuelva true si 2 árboles binarios diferentes A y B tienen el mismo volumen

45. Sea A un árbol en el que cada nodo que no es hoja tiene 2 hijos. Implementar una función para convertir un listado en preorden de A en un listado en postorden.

46. Dada una expresión en postfijo generar un árbol binario asociado, de forma que el recorrido en postorden del mismo devuelva la expresión original

47. Implementar una **función no recursiva** para recorrer un árbol binario en **postorden**

48. Sea A un árbol binario en el que en cada nodo n se guarda un entero correspondiente al número de descendientes propios de n mas uno. Usando el TDA bintree, diseñar una función que dado un nodo n del árbol, calcule la posición de dicho

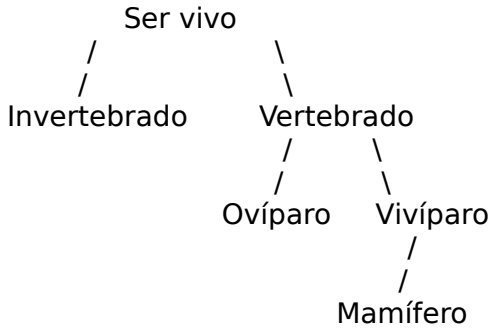
nodo en el listado en inorden de A. La función ha de tener una eficiencia O(p(n)), con p(n) la profundidad del nodo n en A.

49. Implementar una función:

bintree<int>::node siguiente (bintree<int> & A, bintree<int>::node n);

que dado un árbol binario **A,** y un nodo **n** en el árbol, devuelva el nodo siguiente a **n** en el recorrido en **postorden**

50. Supongamos un árbol de clasificación como el de la figura:



De esta forma, un Mamífero es un Vivíparo, que a su vez es un Vertebrado, que a su vez es un Ser Vivo, con lo que transitivamente un Mamífero es un Ser Vivo. Lo contrario no es cierto: un Ser Vivo no es un Mamífero. Por otra parte, un Mamífero no es un Invertebrado ni tampoco un Ovíparo. Se trata de implementar una función que dado un árbol de clasificación genérico nos diga si x es un y . Los elementos del árbol serán cadenas de caracteres. El prototipo para la función sería:

bool esUn (const string& nombre1, const string & nombre2, bintree<string> & A);

que devuelva true si “nombre1” es un “nombre2”. Tener en cuenta que es posible preguntar por cualquier pareja de elementos.

Sugerencia: implementar una función auxiliar:

bool busca (const string & nombre, bintree<string> & A, bintree<string>::node & resultado);
que devuelva tre si “nombre” está en A devolviendo el nodo dónde está.

51. Implementar una función:

bool escorrecta (const bintree<char> & A);

a la que se le pasa un arbol binario completo de char, en cuyas etiquetas pueden aparecer tanto caracteres 'a'..'z' (operandos) como caracteres '+','-','*','/' (operadores) y que devuelva true si es una expresión correcta (los nodos interiores son operadores y las hojas son operandos)

52. Dados dos vectores de etiquetas correspondientes al recorrido en preorden y postorden de un árbol binario de enteros y dos nodos n1,n2 pertenecientes a dicho árbol, implementar una función que devuelva true si n1 es ancestro de n2 y false en caso contrario:

bool es_ancestro (bintree<int> & A, bintree<int>::node n1, bintree<int>::node n2, vector<int> listadopreorden, vector<int> listadopostorden,);

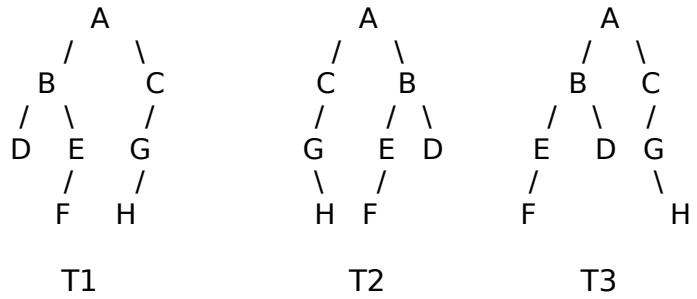
53. Implementar una función:

bool igualContenido (const bintreel<int> &a, const bintre<int> &b);
que, dados dos árboles binarios, devuelva true si ambos contienen la misma información (aunque posiblemente tengan distinta estructura). Es decir, todas y cada una de las etiquetas del árbol a aparecen en alguna rama del árbol b y viceversa.

54. Dados dos árboles binarios, T1 y T2, Implementar una función

bool transformables (bintree<int> & T1, bintree<int> & T2);

que devuelva true si son transformables; es decir, que determine si T1 puede transformarse en T2 intercambiando los hijos izquierdo y derecho de algunos de los nodos de T1. Por ejemplo, los 3 árboles binarios siguientes lo son.



55. 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).

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

bool es_sesgado (const bintree<int> & A);

(b) Implementar una función que transforme un Arbol binario A en un Arbol sesgado hacia la izquierda B. 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.

void transforma_en_sesgado (bintree<int> & A, bintree<int> & B);

56. Dado un arbol binario **A**, y un nodo **n hoja** de dicho árbol implementar una función:

bintree<int>::node siguientehoja (bintree<int> & A, bintree<int>::node n);

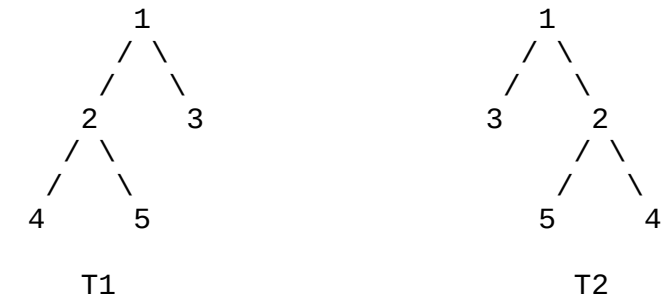
que devuelva la siguiente hoja en el arbol en un recorrido de izquierda a derecha de dichas hojas

57. Sea A un árbol binario en el que cada nodo que no es hoja tiene 2 hijos. Implementar una función para convertir un listado en preorden de A en un listado en postorden.

58. Implementar una función:

bool sonespejo (bintree<int> & T1, bintree<int> & T2);

que devuelva true si dos árboles binarios T1 y `T2 son espejo uno del otro, es decir, que determine si T1 puede transformarse en T2 intercambiando los hijos izquierdo y derecho de todos los nodos de T_1.`
`Ejemplo de árboles espejo:`



59. Diseñar un procedimiento para leer **`un árbol binario de disco`** `usando el recorrido por niveles`

60. mplementar una función para determinar si un árbol binario tiene **exactamente un camino** desde una hoja a la raíz cuya suma de etiquetas sea igual a k.

bool solo_un_suma_k(const bintree<int> &arb, int k)

Consideraciones:

*1.- El reto es **individual***

2.- Los ejercicios que a cada estudiante toca implementar se asignan por la letra de inicio del primer apellido:
2.1.- De la A a la E (ambos incluidos): Elegir 5 ejercicios entre el 1 y el 15
2.2.- De la F a la L (ambos incluidos): Elegir 5 ejercicios entre el 16 y el 30
2.3.- De la M a la R (ambos incluidos): Elegir 5 ejercicios entre el 31 y el 45
2.4.- De la S a la Z (ambos incluidos): Elegir 5 ejercicios entre el 46 y el 60

3.- La solución deberá incluir los códigos de las soluciones y entregarse obligatoriamente en un fichero tar o zip (se sugiere como nombre reto5.tar o reto5.zip)

4.- Las soluciones deberán estar documentadas adecuadamente.

5.- Si se entrega algún código que no compile o no funcione correctamente el reto quedará invalidado.

6.- Si la solución es correcta, se puntuará con 0.2 para la evaluación contínua

7.- El plazo límite de entrega es el 22 de Diciembre a las 23.55h