Estructuras de Datos no Lineales

Práctica 4

Problemas de árboles binarios de búsqueda

TRABAJO PREVIO

Antes de asistir a la sesión de prácticas es obligatorio:

1. **Implementar y probar** el TAD *Abb* representado mediante una estructura dinámica recursiva.
2. Imprimir copia de este enunciado.
3. Lectura profunda del mismo.
4. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
5. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

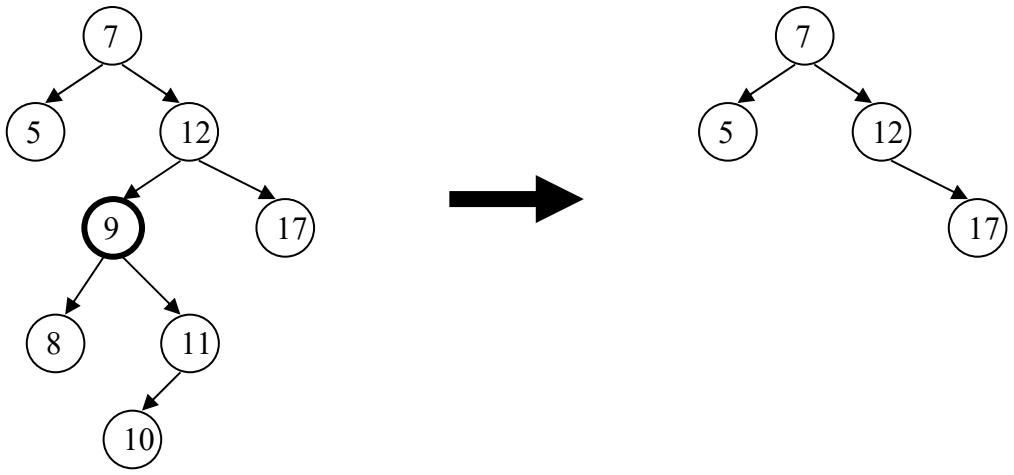
PASOS A SEGUIR

1. Escribir módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
2. Para cada uno de los problemas escribir un programa de prueba, independiente de la representación del TAD elegida, donde se realicen las llamadas a los subprogramas del paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba.

PROBLEMAS

1. Implementa una nueva operación del TAD *Abb* que tomando un elemento del mismo elimine al completo el subárbol que cuelga de él.

Ejemplo: Para el árbol binario de búsqueda de la figura se muestra la transformación si la entrada fuera el valor 9.



2. Un árbol binario de búsqueda se puede equilibrar realizando el recorrido en inorden del árbol para obtener el listado ordenado de sus elementos y a continuación, repartir equitativamente los elementos a izquierda y derecha colocando la mediana en la raíz y construyendo recursivamente los subárboles izquierdo y derecho de cada nodo. Implementa este algoritmo para equilibrar un ABB.

3. Dados dos conjuntos representados mediante árboles binarios de búsqueda, implementa la operación *unión* de dos conjuntos que devuelva como resultado otro conjunto que sea la unión de ambos, representado por un ABB equilibrado.

4. Dados dos conjuntos representados mediante árboles binarios de búsqueda, implementa la operación *intersección* de dos conjuntos, que devuelva como resultado otro conjunto que sea la intersección de ambos. El resultado debe quedar en un árbol equilibrado.

Diferencia simétrica

5. Implementa el operador ♦ para conjuntos definido como $A \blacklozenge B = (A \cup B) - (A \cap B)$. La implementación del operador ♦ debe realizarse utilizando obligatoriamente la operación ∈, que nos indica si un elemento dado pertenece o no a un conjunto. La representación del tipo *Conjunto* debe ser tal que la operación de pertenencia esté en el caso promedio en $O(\log n)$ → Que esté equilibrado

1) 1 - Obtengo el subarbol con buscan (elmto) (Abb2)

2 - Mientras no este vacio hago eliminar (Abb2.elemento(),)

→ Obtengo la raíz de ese subarbol

2) Obtengo el inorden. HAY QUE USAR EL VECTOR STL → PUSH_BACK() → Inserto al final
→ SIZE() → Da el tamaño

Usando una lista in orden de los elementos

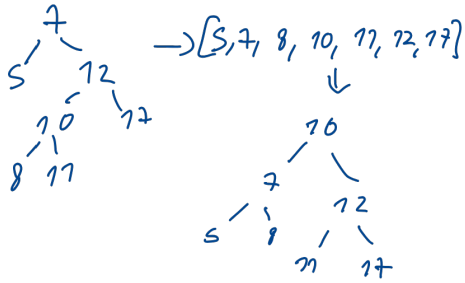
```
inorden Abb RecCAbb &lista)
  if (!Abb.vacia()){
     AbbRec(Abb.izqdo(), lista);
     lista.insertar(Abb.elemento(), lista.fin());
     Abb Rec(Abb.dcho, lista);
  }
```

Cuando reciba un vector de tam 1 o 2 lo inserto directamente

COMO PIDE MODIFICARLO, CREO EL Abb equilibrado Y LO ASIGNO AL DE ENTRADA

hago una función que divida la lista en 2, divida Lista (lista, Lista1, Lista2)

```
Abb equilibrar Abb Rec (Abb & A, Lista: eltos)
  if (!eltos.vacia()) {
    A.insertar (elemento del medio de la lista);
    lista1; lista2
    divida lista (elto, lista1, lista2)
    equilibrar Rec (A, lista1)
    equilibrar Rec (A, lista 2)
  }
```



3 union Abb (Abb &A, Abb &C)

```
if (!A.vacio()){
    C.inserta(A.elemento());
    rellena_Abb(A.izqdo(), C)
    rellena_Abb(A.drcho(), C)
}
```

4 interseccion Abb (Abb &A, Abb &C, Abb &Inter)

```
if (!A.vacio()){
    if (!C.buscar(A.elemento).vacio()){
        Inter.inserta(A.elemento());}
    interseccion_Abb(A.izqdo(), C, Inter);
    interseccion_Abb(A.drcho(), C, Inter);
}
```

5 operacion Abb (Abb &A, Abb &C, Abb &R)

```
if (!A.vacio()){
    if (C.buscar(A.elemento).vacio()){
        R.inserta(A.elemento());}
    rellena_Abb(A.izqdo(), C)
    rellena_Abb(A.drcho(), C)
}
```

2 Es modificar el TAD, hay que quitar buscar const, obtengo el Abb que contiene el elemento, llamo al destructor y redirecciono el puntero. Al modificar buscar tengo que tener el Abb del padre. Busco el Abb, con so z hago delete z y despues z=nullptr;

3 Hay que hacer un TAD Conjunto en la parte privada pongo un Abb siempre equilibrado → El cstructor recibe un Abb y llama una func privada equilibrar

↳ Es como hacer el set de la std

Sobrecargo un operador para representar la unión, intersección y las operaciones devuelven un TAD conjunto.

En la diferencia simétrica hay que recorrer dos veces el árbol