Resumen de Clases **Baja en ABB**

Idea

La función de Baja o Suprime en un ABB tiene que operar de tal manera que después de practicada el árbol siga siendo de búsqueda (ordenado menores descendientes por izquierda y mayores descendientes por derecha).

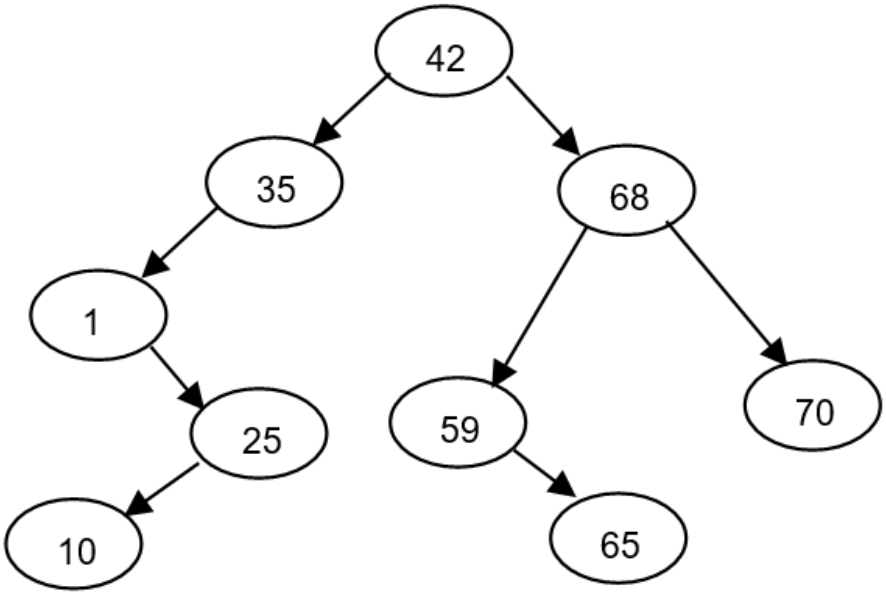
Situaciones:

Caso 1: Analizaremos primero dar de baja un nodo hoja. En este caso solo practicamos la baja y el árbol sigue siendo de búsqueda. Ej nodo 10 o 65 o 70.

Caso 2: Analizaremos ahora eliminar un nodo que solo tenga descendencia por izquierda, caso del nodo 25 o el 35. Si tomamos el 25, debemos asignar su hijo izquierdo(10) al padre de 25(1) en el puntero izq o der según corresponda.

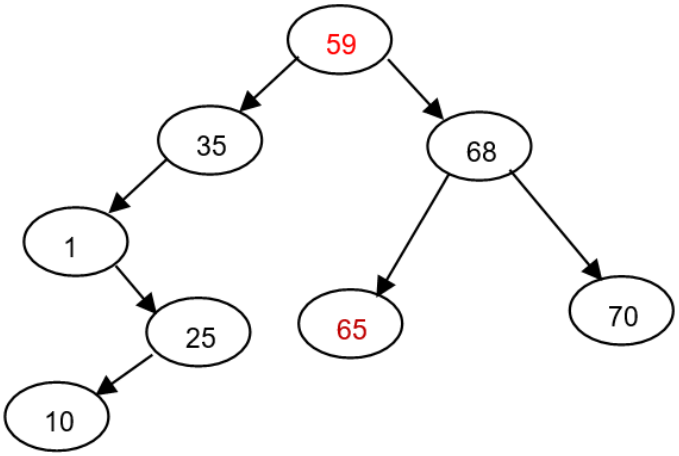
Caso 3: Eliminar un nodo que solo tenga descendencia por derecha. Caso del 59, entonces, debemos asignar su hijo derecho(65) al padre de 59(68) en el puntero izq o der según corresponda.

Caso 4: El caso más complicado es eliminar un nodo que tenga descendencia tanto por derecha como por izquierda. Ej 42 o 68.
Para este caso, para no alterar el orden lexicográfico, debemos buscar el menor de los descendientes por izquierda y promover este nodo (su información) en reemplazo del nodo a eliminar, luego quitar el nodo menor descendiente por izquierda aplicando caso 2 o 3 según corresponda.



De esta manera, si en el caso anterior eliminamos el 42, tomamos el menor del subárbol derecho (raíz en 68) lo que nos da 59 y reemplazamos 59 por 42, luego eliminamos el 59 promoviendo el reemplazo por el 65, el árbol nos queda como se muestra a continuación.

Nota: el menor de un ABB es el primer nodo sin hijo izquierdo en un descenso siempre por el link izquierdo.



Implementación en C++

La implementación se puede diseñar en dos funciones:

- abinario_ptr_menor: busca el puntero al menor de un árbol y también su padre (parámetro pasado por referencia).
- Abinario_suprime: realiza la baja, usando la función anterior.

```
NABinario * abinario_ptr_menor(NABinario* arbol,NABinario * &padre) {
   if (arbol == NULL) return NULL;
   if (arbol->iz == NULL)  // entonces arbol apunta al elemento mas pequeño
            return arbol;
    else  { // el nodo apuntado por arbol tiene un hijo izquierdo que indudablemente es menor
             padre = arbol;  // asignacion necesaria para cambiar parametro pasado por referencia
             return abinario_ptr_menor(arbol->iz,arbol);
    }
}

void abinario_suprime (NABinario* &arbol, int dato_a_eliminar) {
     if (arbol != NULL) {
        if (dato_a_eliminar < arbol->dato) abinario_suprime(arbol->iz, dato_a_eliminar);
         else if (dato_a_eliminar > arbol->dato) abinario_suprime(arbol->de, dato_a_eliminar);
          else if (arbol->iz == NULL && arbol->de == NULL) arbol = NULL;
           /*si se llega aqui, dato_a_eliminar es el nodo apuntdo por arbol*/
            else if (arbol->iz == NULL && arbol->de == NULL)
               arbol = NULL; //suprime la hoja que contiene dato_a_eliminar
              else if (arbol->iz == NULL)    arbol = arbol->de;
               else if (arbol->de == NULL)   arbol = arbol->iz;
                 else   {  NABinario * padremenxder = arbol;
                           NABinario * ptrmenorxder = abinario_ptr_menor(arbol->de,padremenxder);

             if (padremenxder == NULL) cout << "padreNulo" << endl;
               else cout << "padreeee:" << padremenxder->dato << endl;
                           arbol->dato = ptrmenorxder->dato;
                           if (padremenxder->iz == ptrmenorxder)
                                  padremenxder->iz = ptrmenorxder->de;
                             else padremenxder->de = ptrmenorxder->de;
                       }
     }
}
```