



Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 9

Curso 2022-2023

Marta N. Gómez



T3. Tipos Abstractos de Datos (TAD)

- Árboles.
 - Conceptos generales
 - Realización del TAD Árbol Binario
 - Recorridos de Árboles Binarios
 - Árboles Binarios de Búsqueda (ABB)
 - Árboles Equilibrados (AVL)
 - Montículos

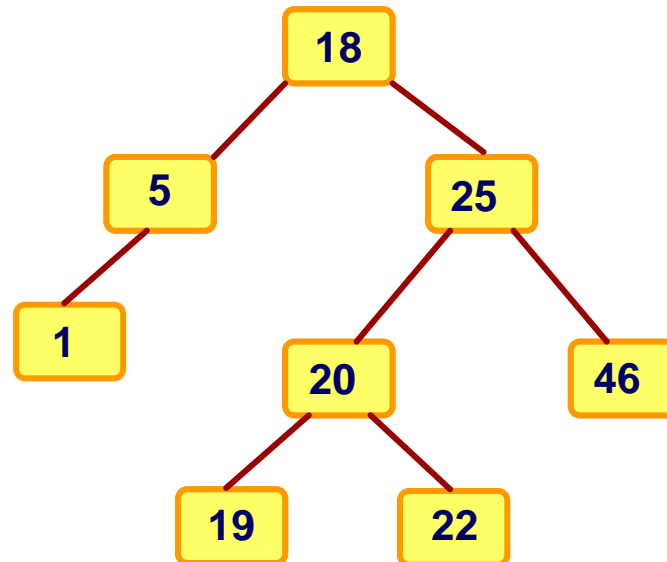


T3. Tipos Abstractos de Datos (TAD)

- **Árboles.**
 - Conceptos generales
 - Realización del TAD Árbol Binario
 - Recorridos de Árboles Binarios
 - **Árboles Binarios de Búsqueda (ABB)**
 - Árboles Equilibrados (AVL)
 - Montículos

ÁRBOLES BINARIOS DE BÚSQUEDA (ABB)

Un **Árbol binario de búsqueda (ABB)** es aquel que dado un nodo, todos los datos del **subárbol izquierdo** son **menores** que el dato de dicho nodo y los datos del **subárbol derecho** son **mayores o iguales** que el dato de dicho nodo.



```
//-----Clase Nodo
class Nodo {
    public:
        CData dato;
        shared_ptr<Nodo> hizq = nullptr;
        shared_ptr<Nodo> hdch = nullptr;
    public:
        Nodo(const CData& d):dato{d} {};

        const CData &getDato() const;
        void setDato(const CData &newDato);

        const shared_ptr<Nodo> &getHizq() const;
        void setHizq(const shared_ptr<Nodo> &newHizq);

        const shared_ptr<Nodo> &getHdch() const;
        void setHdch(const shared_ptr<Nodo> &newHdch);

        void procesarNodo () const;
};
```

```
//-----Clase Arbol Binario Busqueda
class Arbol {
private:
    shared_ptr<Nodo> raiz = nullptr;
public:
    Arbol():raiz(nullptr){};
    Arbol(CDato const &dato);

    bool empty() const;

    void addHizq(Arbol const &Ai);
    void addHdch(Arbol const &Ad);
    void construirArbol (Arbol const &Ai, Arbol const &Ad, CDato const &dato);

    CDato const &getDatoNodo () const;
    const shared_ptr<Nodo> &getHiNodo() const;
    const shared_ptr<Nodo> &getHdNodo() const;

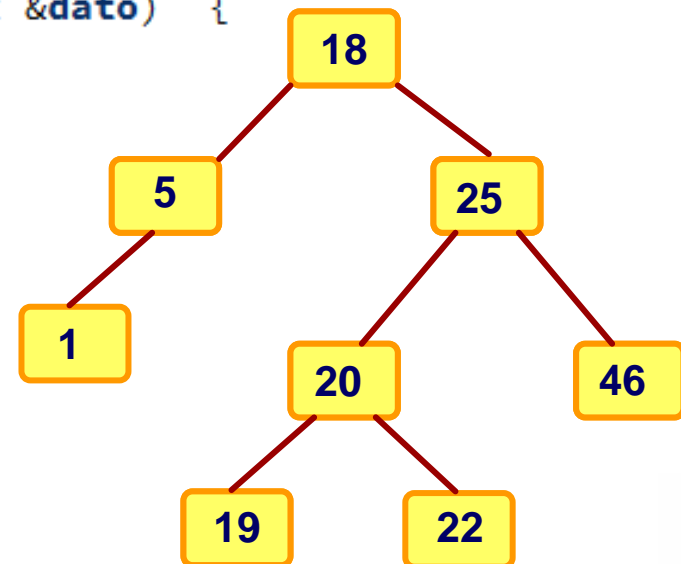
    const shared_ptr<Nodo> &getRaiz() const;
    void setRaiz(const shared_ptr<Nodo> &newRaiz);
}
```

```

// Insertar nodos en el árbol BB recursivamente
void Arbol::insertadoNodoABBRec(shared_ptr<Nodo> &A, CDato const &dato) {
    if (A == nullptr) {
        A = make_shared<Nodo>(Nodo{dato});
    }
    else {
        if (A->getDato().getN() > dato.getN()) {
            insertadoNodoABBRec(A->hizq, dato);
        }
        else if (A->getDato().getN() <= dato.getN()) {
            insertadoNodoABBRec(A->hdch, dato);
        }
    }
}

void Arbol::insertarNodoABBRec(CDato const &dato) {
    insertadoNodoABBRec(raiz, dato);
}

```

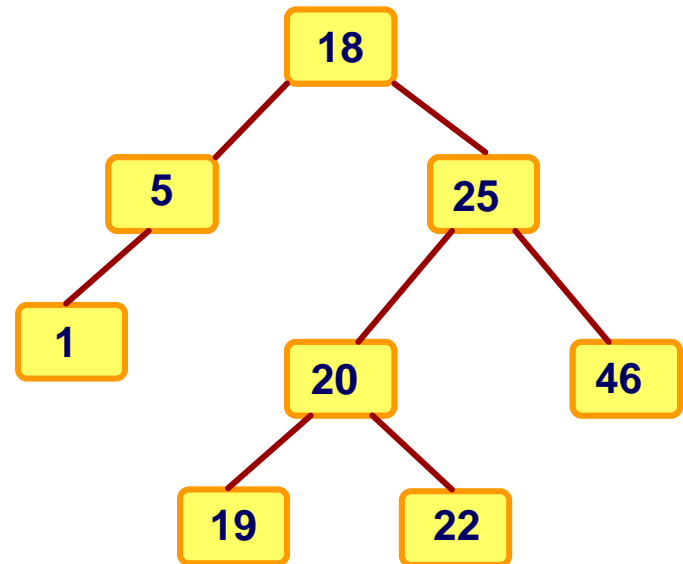


ÁRBOLES ABB

```

// Insertar nodos en el árbol BB iterativamente
void Arbol::insertarNodoABBIter(CDato const &dato) {
    shared_ptr<Nodo> Anew(make_shared<Nodo>(Nodo{dato}));
    if (raiz == nullptr) {
        raiz = Anew;
    }
    else {
        shared_ptr<Nodo> Ant, A = raiz;
        // Bucle para buscar donde se añade el nuevo nodo
        while (A != nullptr) {
            Ant = A; // puntero que guarda la dirección del nodo padre
            if (A->getDato().getN() > dato.getN()) {
                A = A->hizq;
            } else {
                A = A->hdch;
            }
        }
        // Se inserta el nuevo nodo
        if (Ant->getDato().getN() > dato.getN()) {
            Ant->hizq = Anew;
        } else {
            Ant->hdch = Anew;
        }
    }
}

```



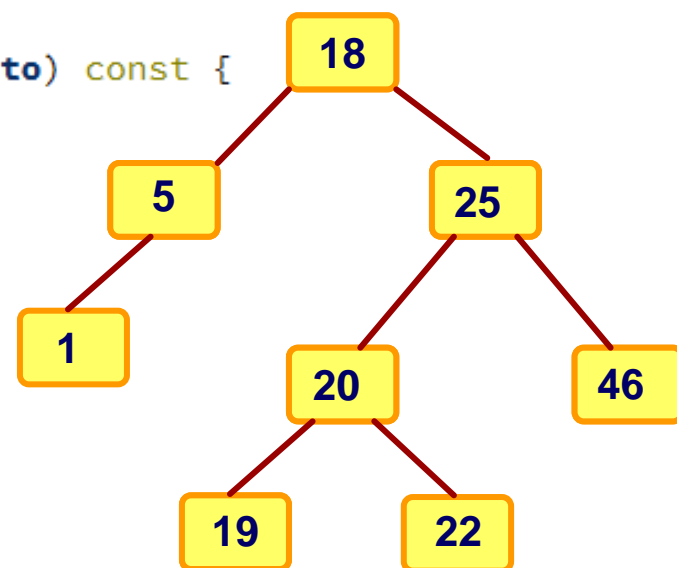
ÁRBOLES ABB


```
// Buscar nodos en el árbol BB recursivamente
shared_ptr<Nodo> Arbol::busquedaNodoABBRec(shared_ptr<Nodo> A, CDato const &dato) const {
    CDato e;

    if (A != nullptr) {
        if (A->getDato().getN() > dato.getN()) {
            A = busquedaNodoABBRec(A->hizq, dato);
        }
        else if (A->getDato().getN() < dato.getN()) {
            A = busquedaNodoABBRec(A->hdch, dato);
        }
    }

    return A;
}

shared_ptr<Nodo> Arbol::buscarNodoABBRec(CDato const &dato) const {
    return busquedaNodoABBRec(raiz, dato);
}
```



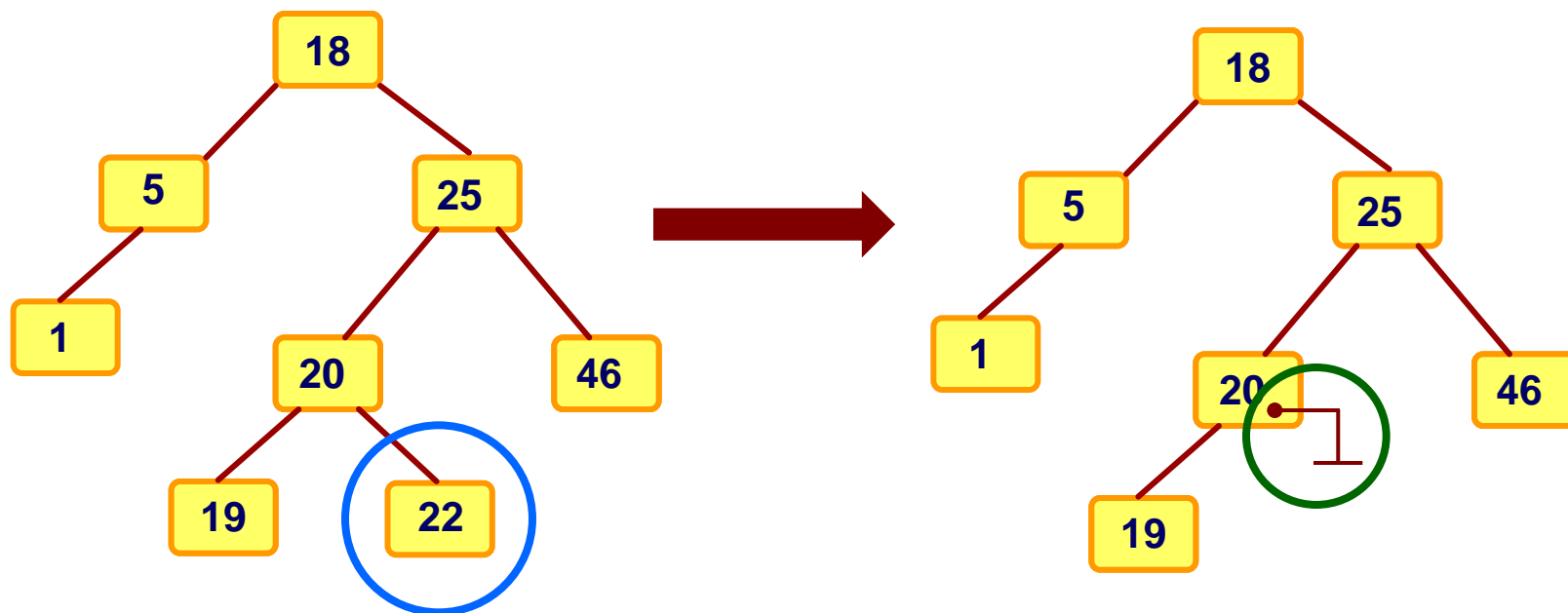
ELIMINACIÓN:

Operación más complicada porque puede implicar la reorganización de varios nodos del árbol.

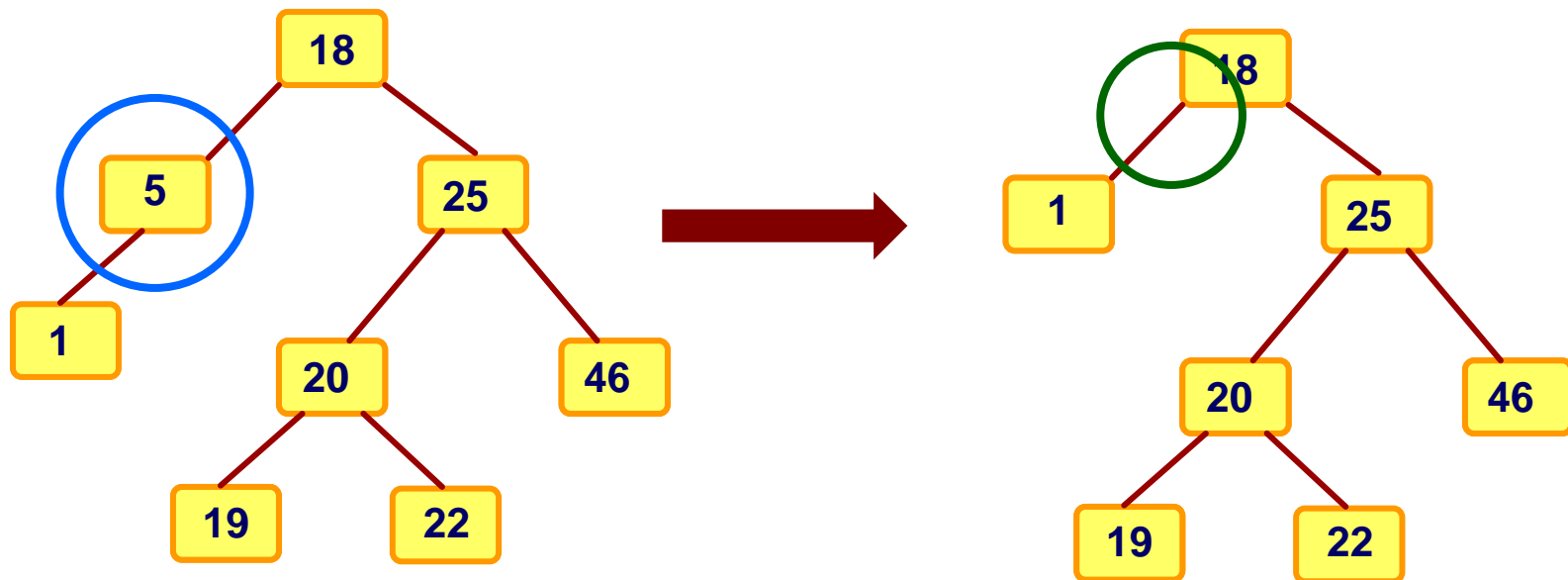
Los pasos a seguir son:

- Buscar el nodo a eliminar, sin perder la posición del nodo padre.
- Casos:
 - **Nodo hoja:** asignar al nodo padre el valor nulo y liberar el nodo eliminado.
 - **Nodo con un único descendiente:** asignar al enlace del nodo padre el descendiente del nodo a eliminar.
 - **Nodo con dos descendientes:** reemplazar el nodo a eliminar por el nodo más a la derecha del subárbol izquierdo (**el mayor de los menores**) o por el nodo más a la izquierda del subárbol derecho (**el menor de los mayores**) y eliminar el nodo.

Eliminar un Nodo HOJA



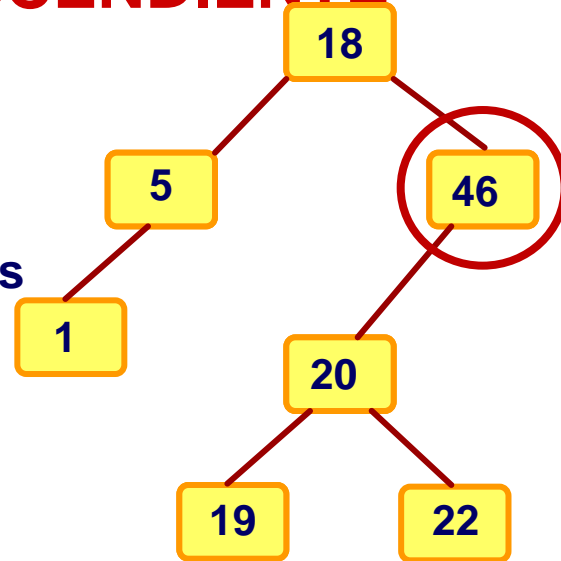
Eliminar un Nodo Con UN DESCENDIENTE



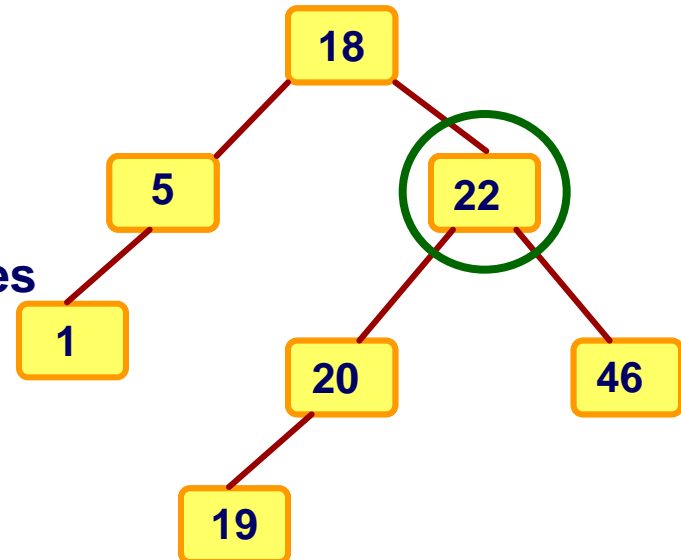
Eliminar un Nodo Con DOS DESCENDIENTE

ÁRBOLES

Opción 1:
menor de los mayores



Opción 2:
mayor de los menores



```
void Arbol::eliminarNodoABB(CDato dato) {

    shared_ptr<Nodo<CDato>> A = raiz;
    shared_ptr<Nodo<CDato>> Apadre = nullptr;
    bool enc {false};

    // Buscamos el nodo que se quiere borrar y
    // se guarda la referencia del nodo padre en Apadre
    while ((A != nullptr) && !enc) {
        if (A->dato.getN() == dato.getN()) {
            enc = true;
        }
        else {
            Apadre = A;
            if (dato.getN() < A->dato.getN()) {
                A = A->hizq;
            }
            else {
                A = A->hdch;
            }
        }
    }
}
```

```
// El proceso continua siempre que se encuentre el dato
if (A != nullptr) {

    // El nodo tiene los dos hijos,
    // luego se sustituye por el mayor de los menores
    if (A->hizq != nullptr && A->hdch != nullptr) {
        A->dato = elMAYORdeMENORESIter(A->hizq);

        // Se busca el nodo con el mayor de los menores
        Apadre = A;
        A = A->hizq;
        while (A->hdch != nullptr) {
            Apadre = A;
            A = A->hdch;
        }
    }
}
```

```
// El nodo No tiene los dos descendiente
// También incluye el borrado del mayor de menores
shared_ptr<Nodo<CDato>> Anext;
if (A->hizq != nullptr) {
    Anext = A->hizq;
}
else {
    Anext = A->hdch;
}

// Se actualiza el nodo padre del nodo borrado
if (Apadre == nullptr) {
    // Cuando se borra el nodo raiz del árbol
    raiz = Anext;
}
else {
    if (Apadre->hizq == A) {
        Apadre->hizq = Anext;
    }
    else {
        Apadre->hdch = Anext;
    }
}
}
}
```


Busca el menor de los nodos mayores recursivamente

```
const CDato& Arbol::eLMAYORdeMENORES(shared_ptr<Nodo<CDato>> A) {  
    if (A->hdch != nullptr) {  
        return eLMAYORdeMENORES(A->hdch);  
    }  
    else {  
        return A->dato;  
    }  
}
```

Busca el menor de los nodos mayores iterativamente

```
const CDato& Arbol::elMAYORdeMENORESIter (shared_ptr<Nodo<CDato>> A)
{
    while (A->hdch != nullptr) {
        // Bajamos por la rama derecha
        A = A->hdch;
    }
    return A->dato;
}
```