Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 10

Curso 2023-2024

Marta N. Gómez

UNIVERSIDAD NEBRIJA

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

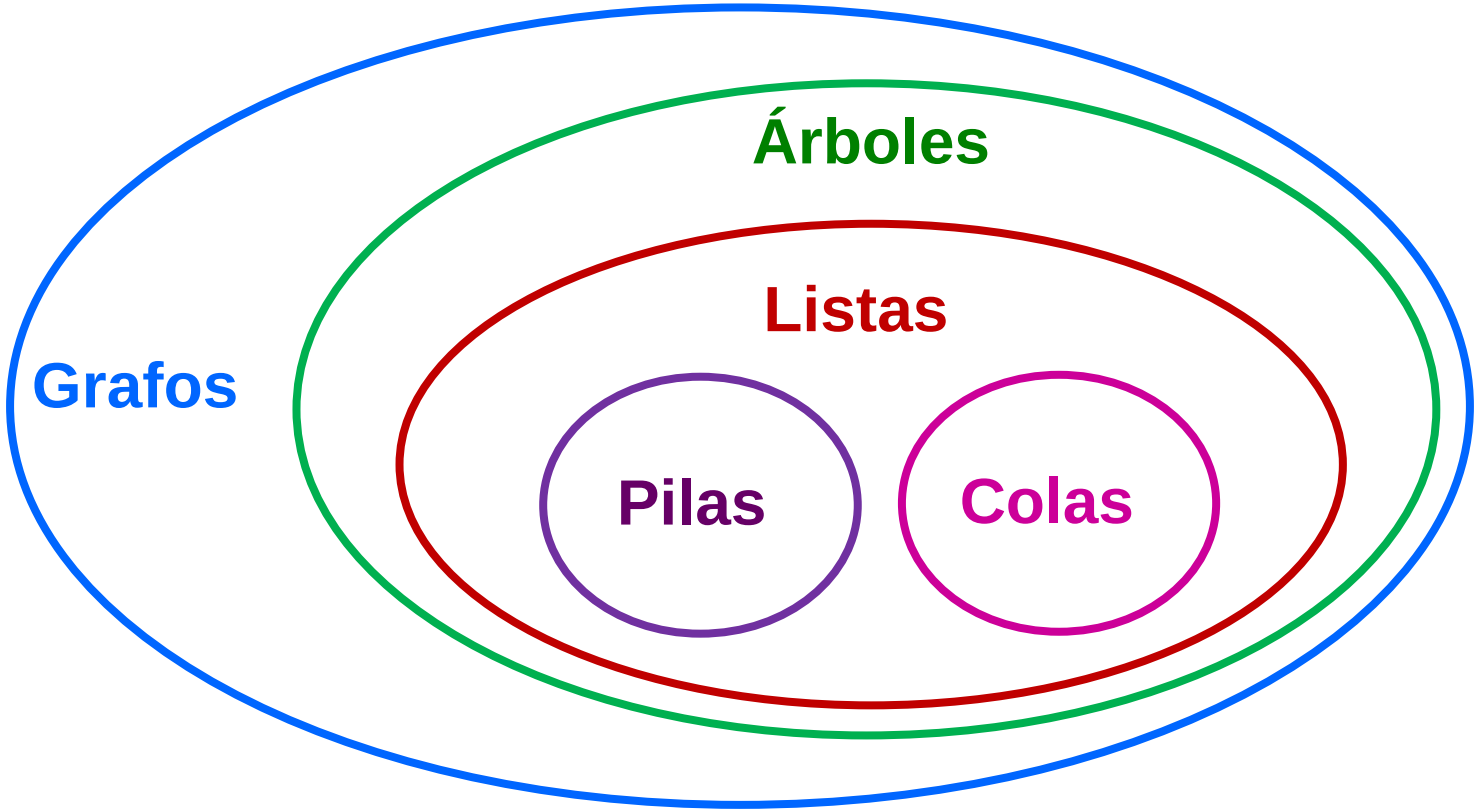
T3. Tipos Abstractos de Datos (TAD)

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

UNIVERSIDAD NEBRIJA

ÍNDICE

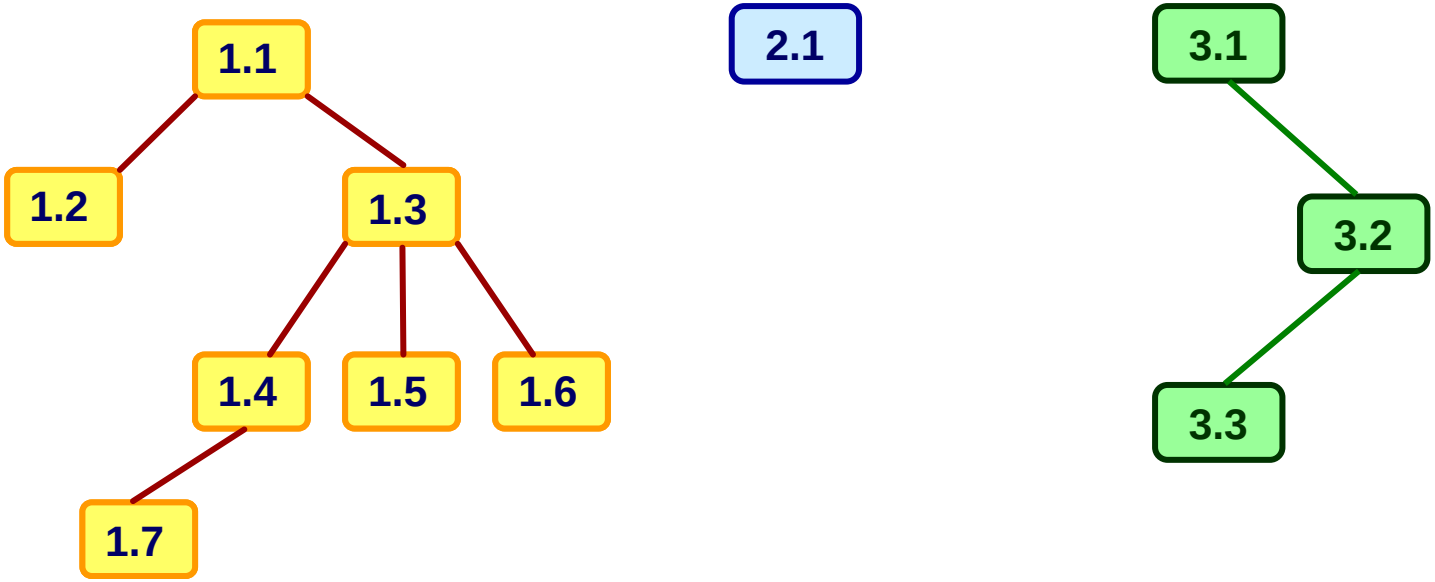
ÁRBOLES-Conceptos Generales



Un **Árbol** es una estructura de datos **no lineal**.

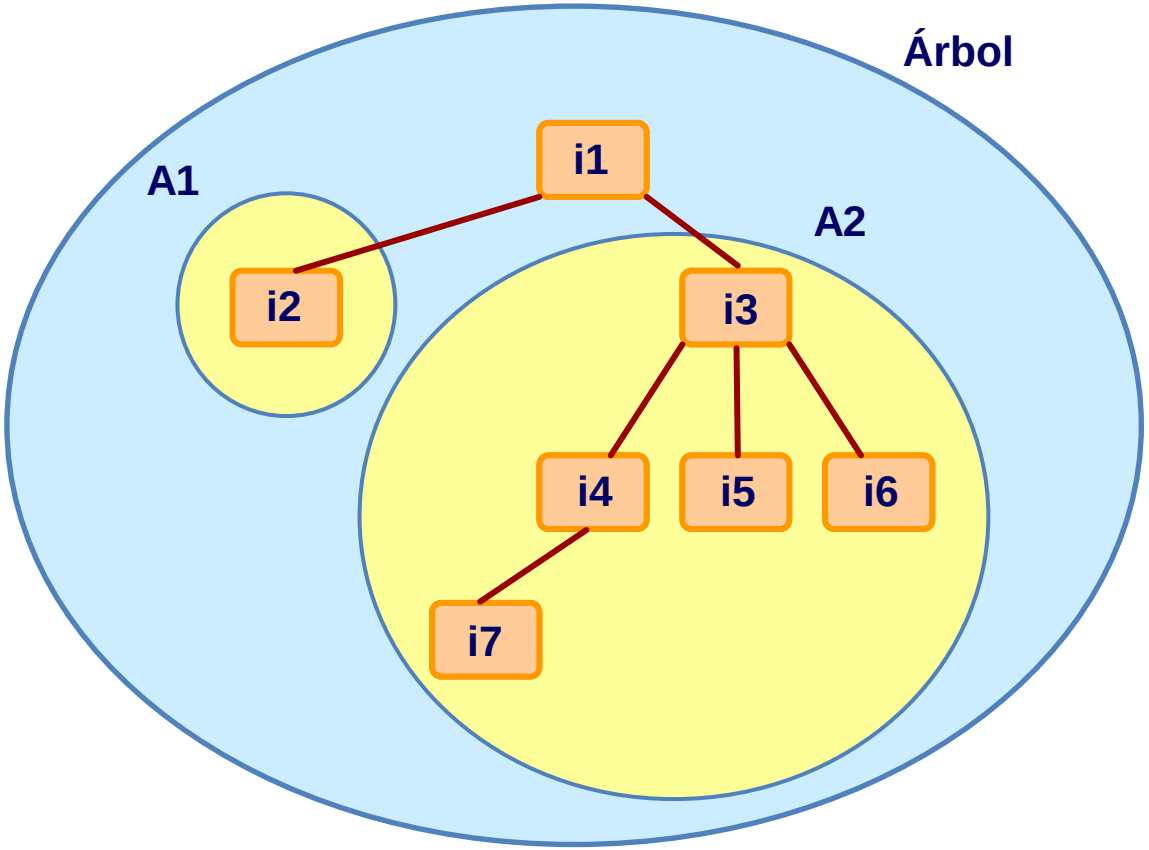
Su organización es **jerárquica** y se compone de elementos **homogéneos**.

Cada elemento puede tener **varios elementos sucesores**, pero **un único elemento antecesor**.

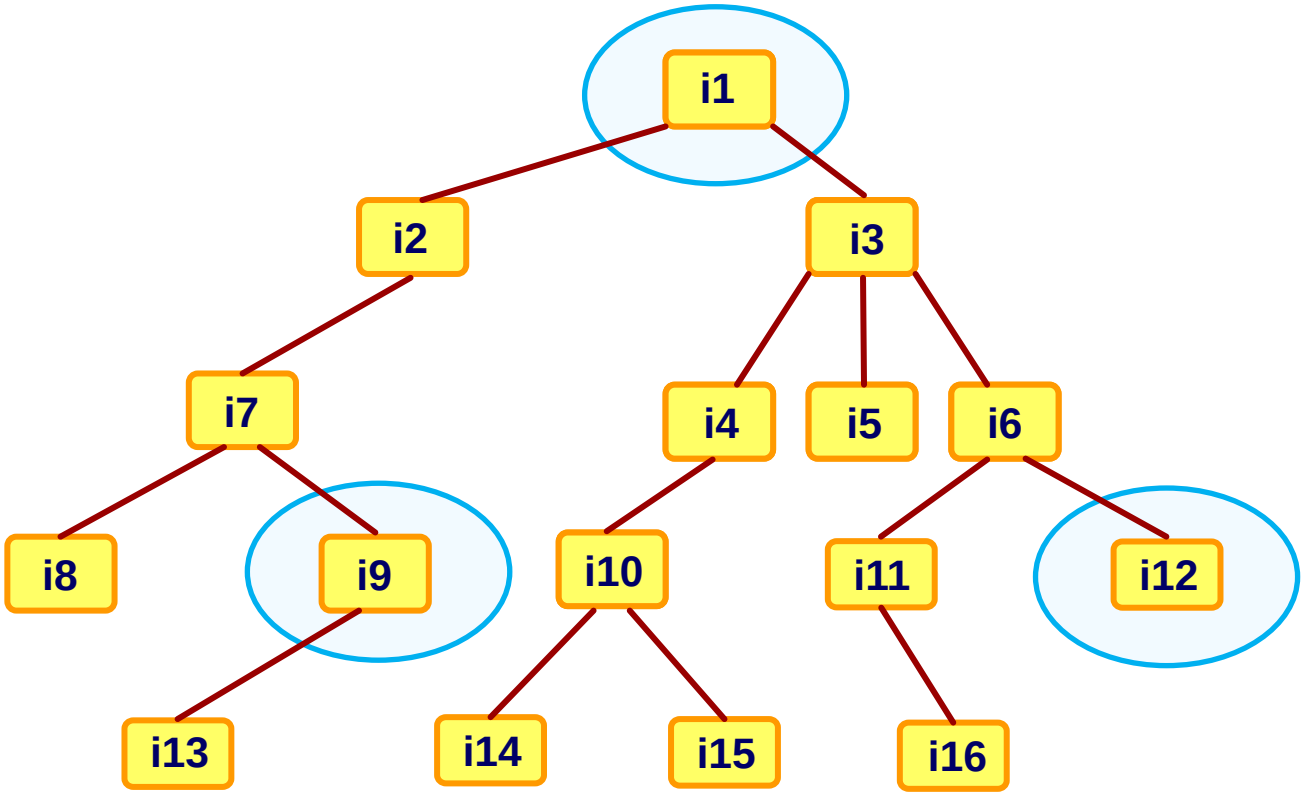


Definición recursiva de **Árbol:**

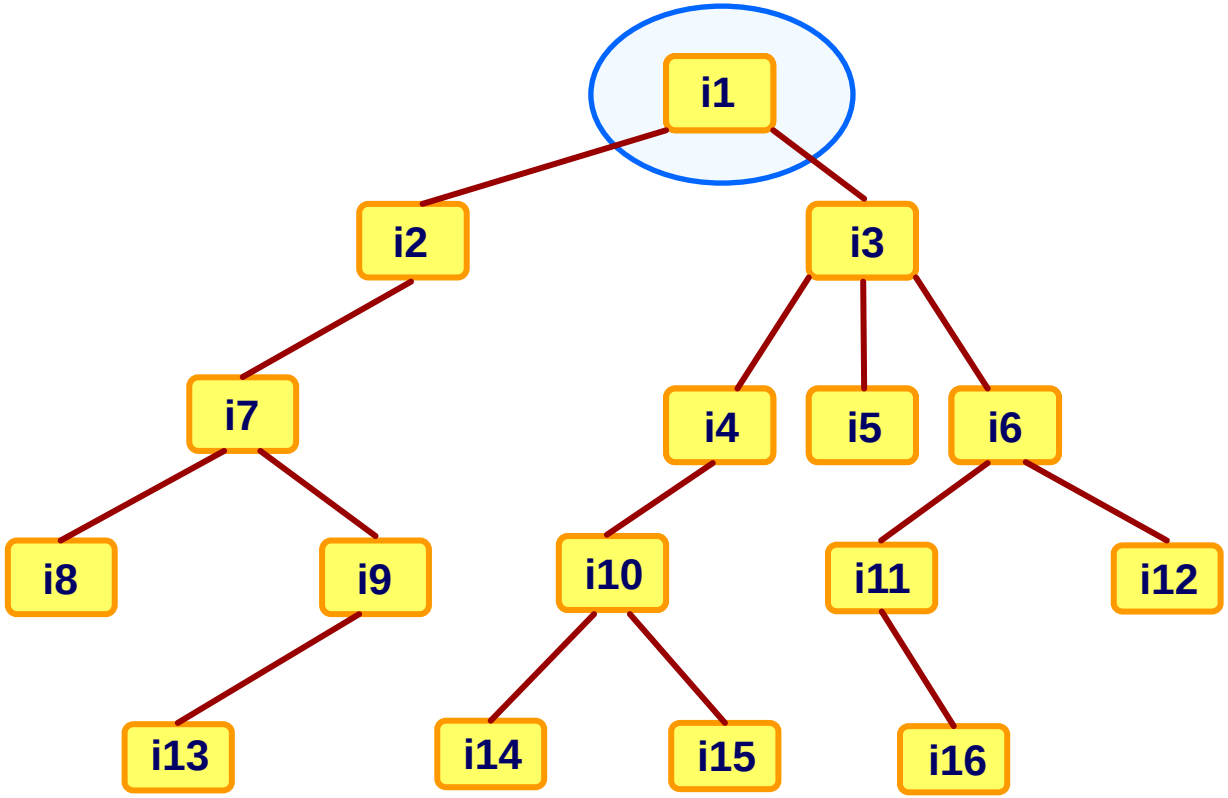
- árbol **vacío**
- un elemento con cierta información y una serie de **árboles**, A1,..., An, **disjuntos**.



Nodo

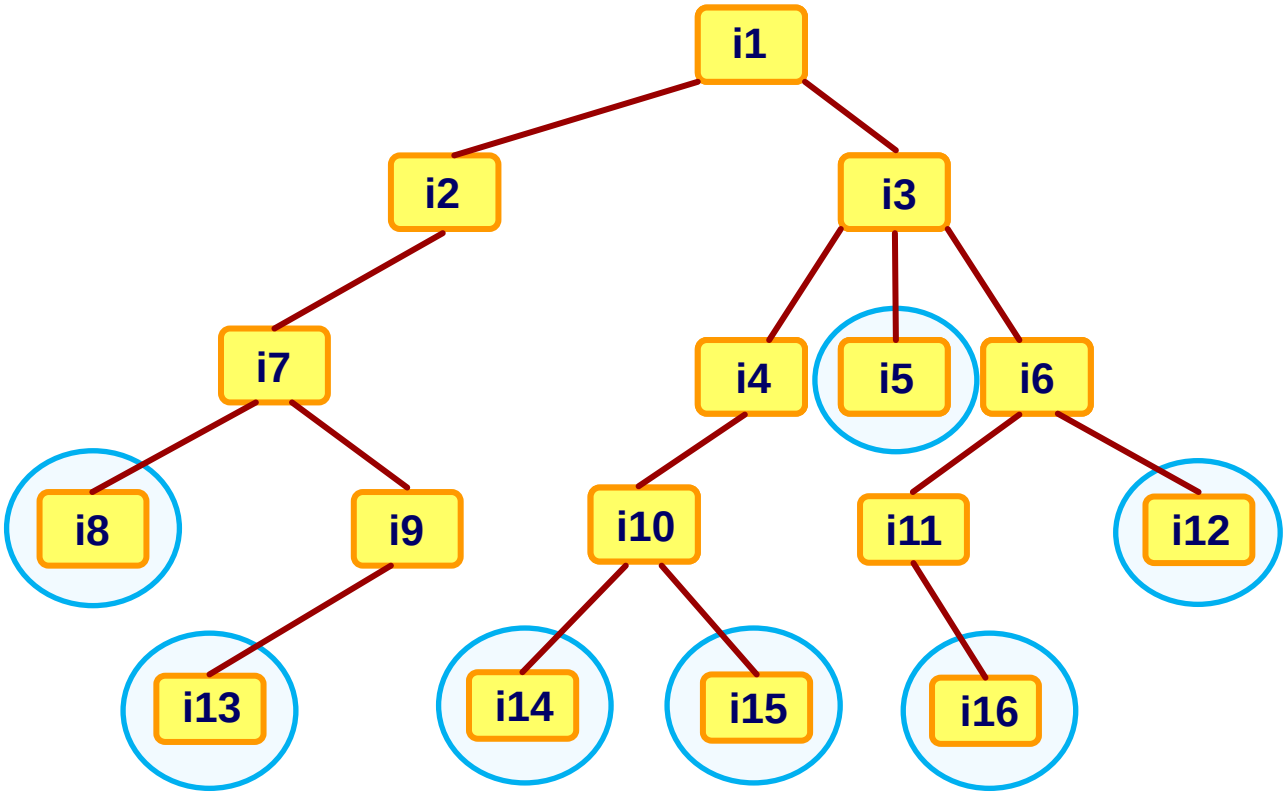


Nodo Raíz: primer nodo o elemento del árbol.



Nodo Raíz

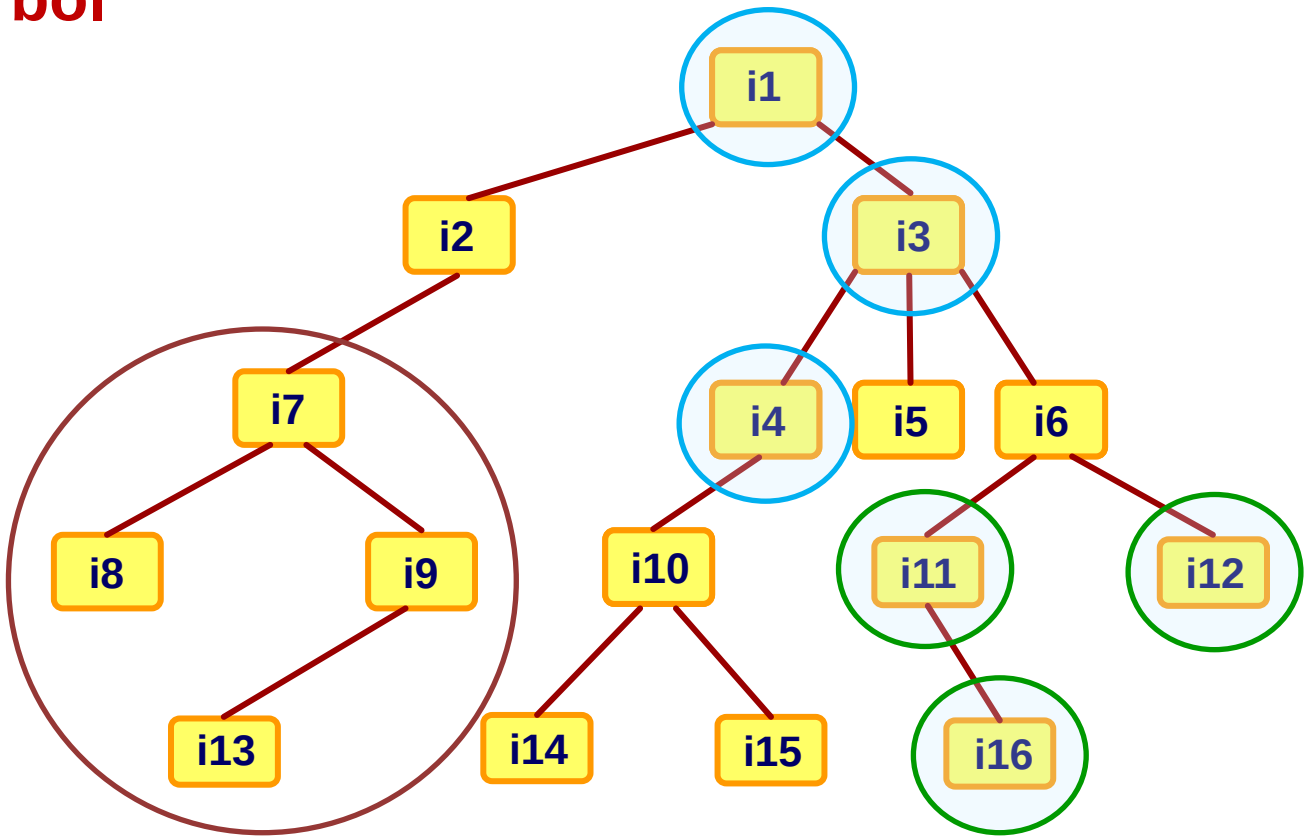
Nodo Hoja: nodo terminal o que no tiene hijos



Ascendientes: son los nodos del camino que van desde el nodo raíz hasta dicho nodo. Para i10 son: i1, i3, i4.

Descendientes: son todos los nodos accesibles por un camino que comience en dicho nodo. Para i6 son: i11, i12, i16.

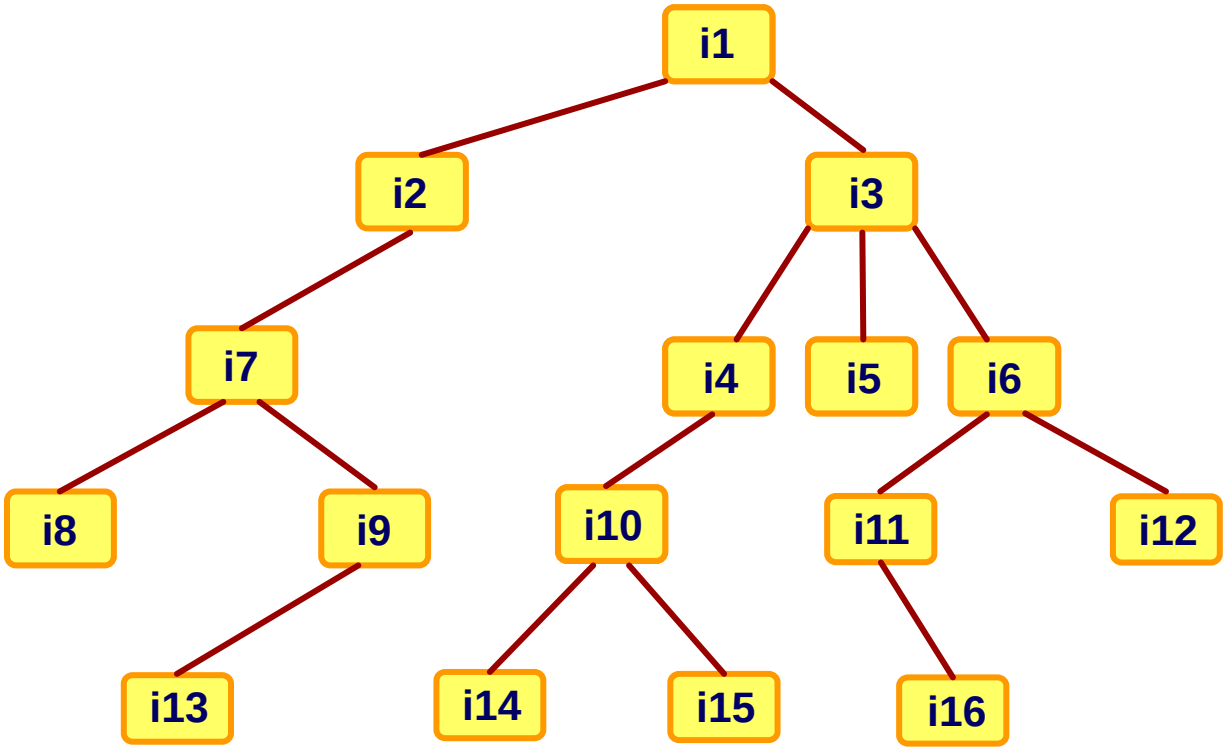
Subárbol



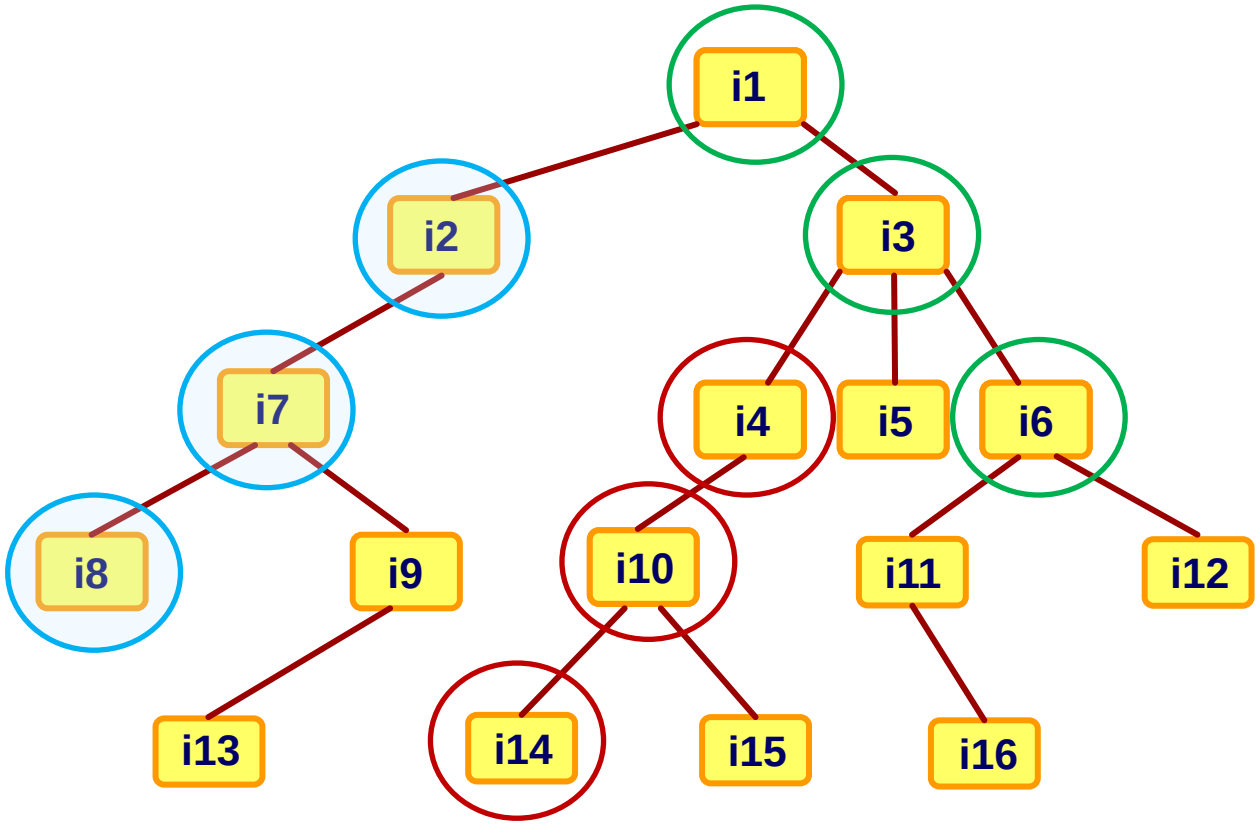
Nodo Padre

Nodo Hijo: nodos raíz de los subárboles.

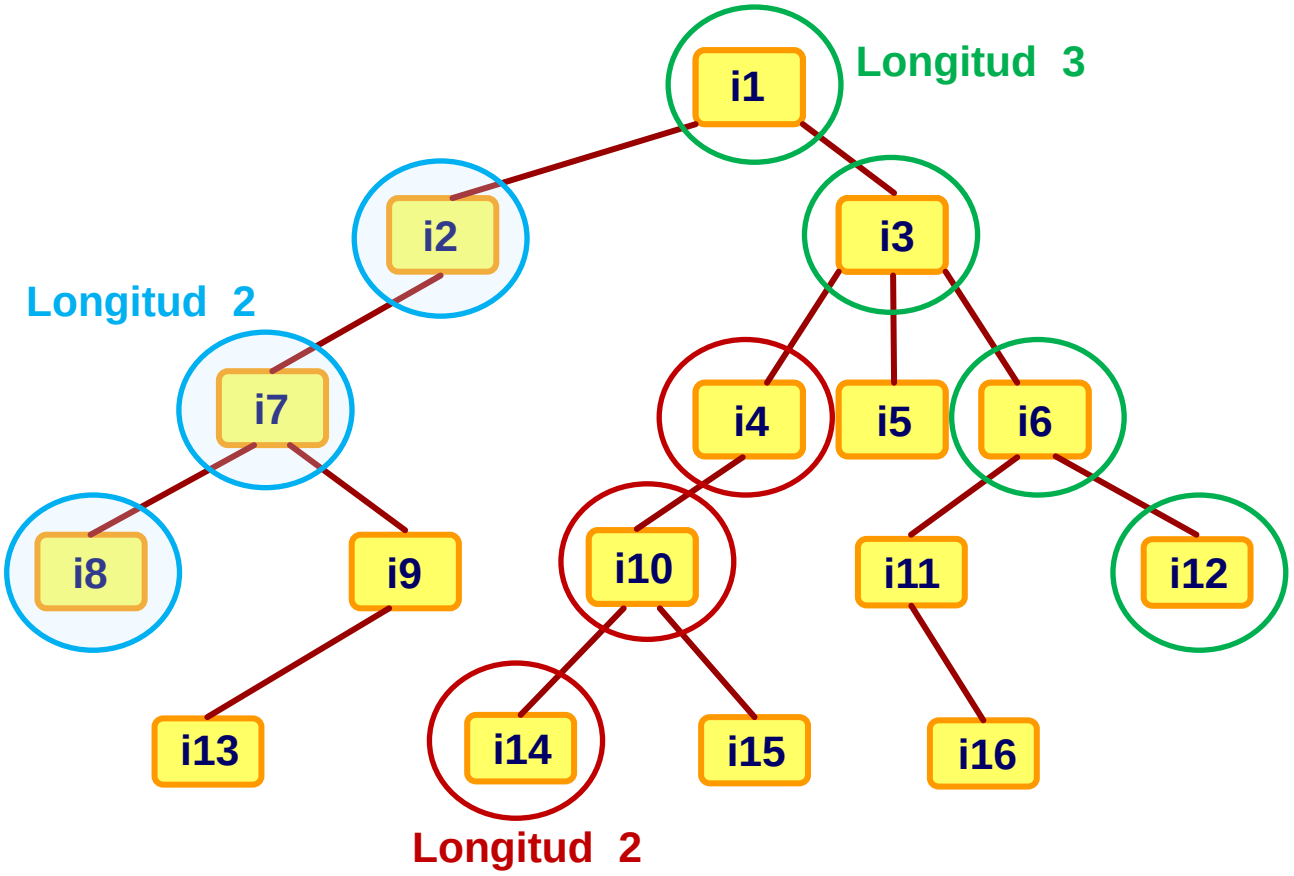
Nodo Interno: nodo con algún hijo.



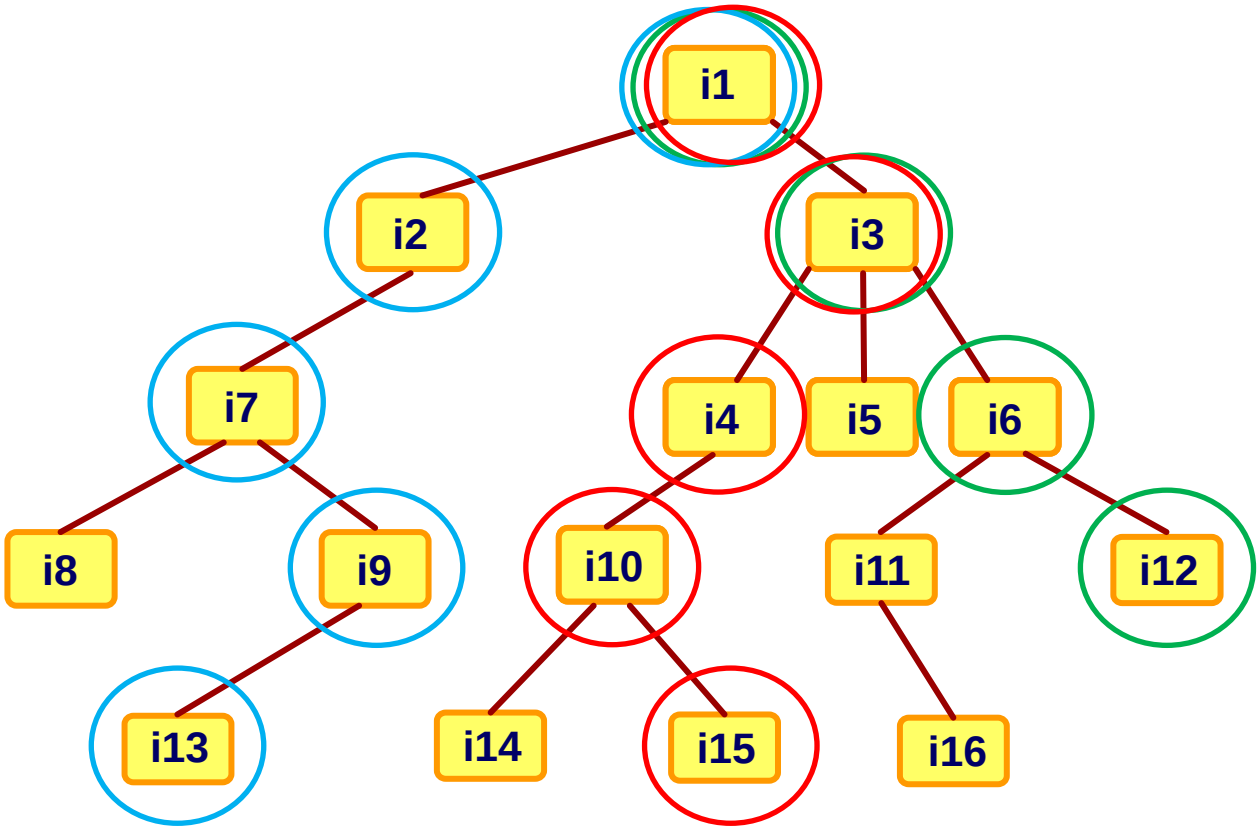
Camino: secuencia de nodos consecutivos donde **cada nodo** es el **padre** del **siguiente nodo** de la secuencia.



La **longitud de un camino** es el **número de nodos del camino menos uno**.



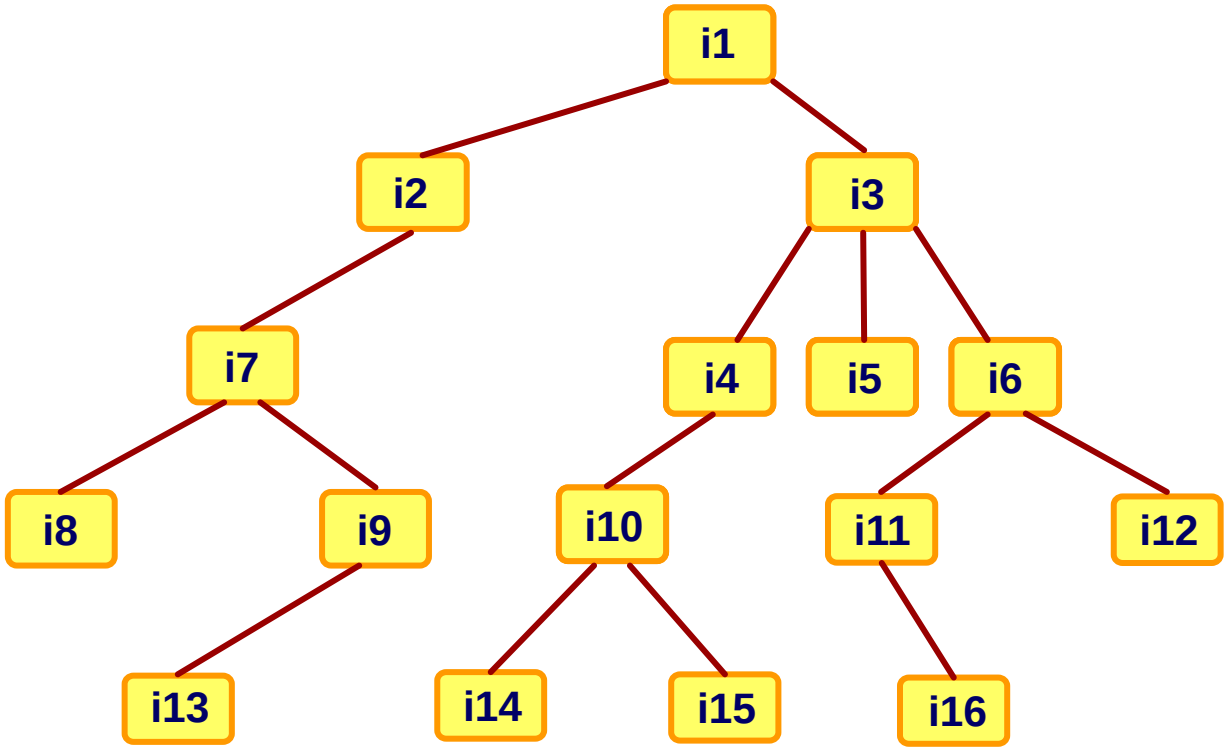
Rama: Camino desde la raíz hasta un nodo hoja



Nivel de un NODO: número de nodos desde la raíz hasta dicho nodo.

Altura de un NODO: longitud del camino más largo desde el nodo hasta una hoja.

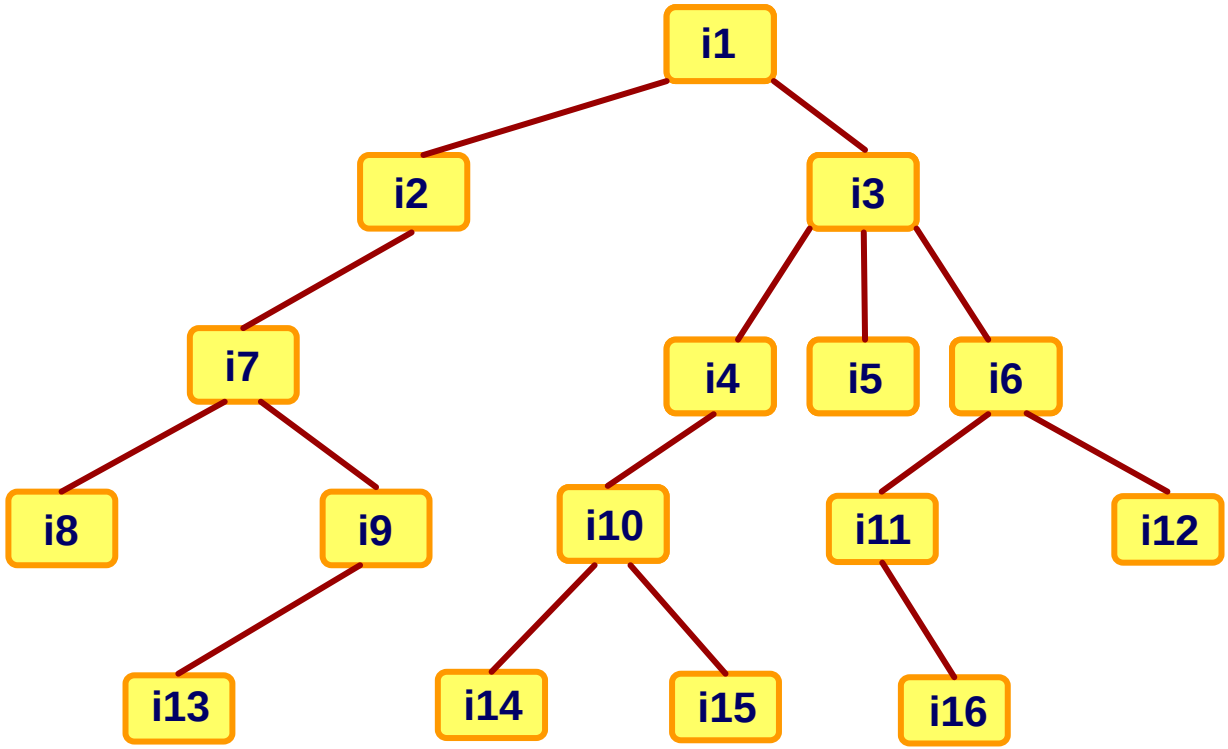
Grado de un NODO: **número de hijos** de dicho nodo



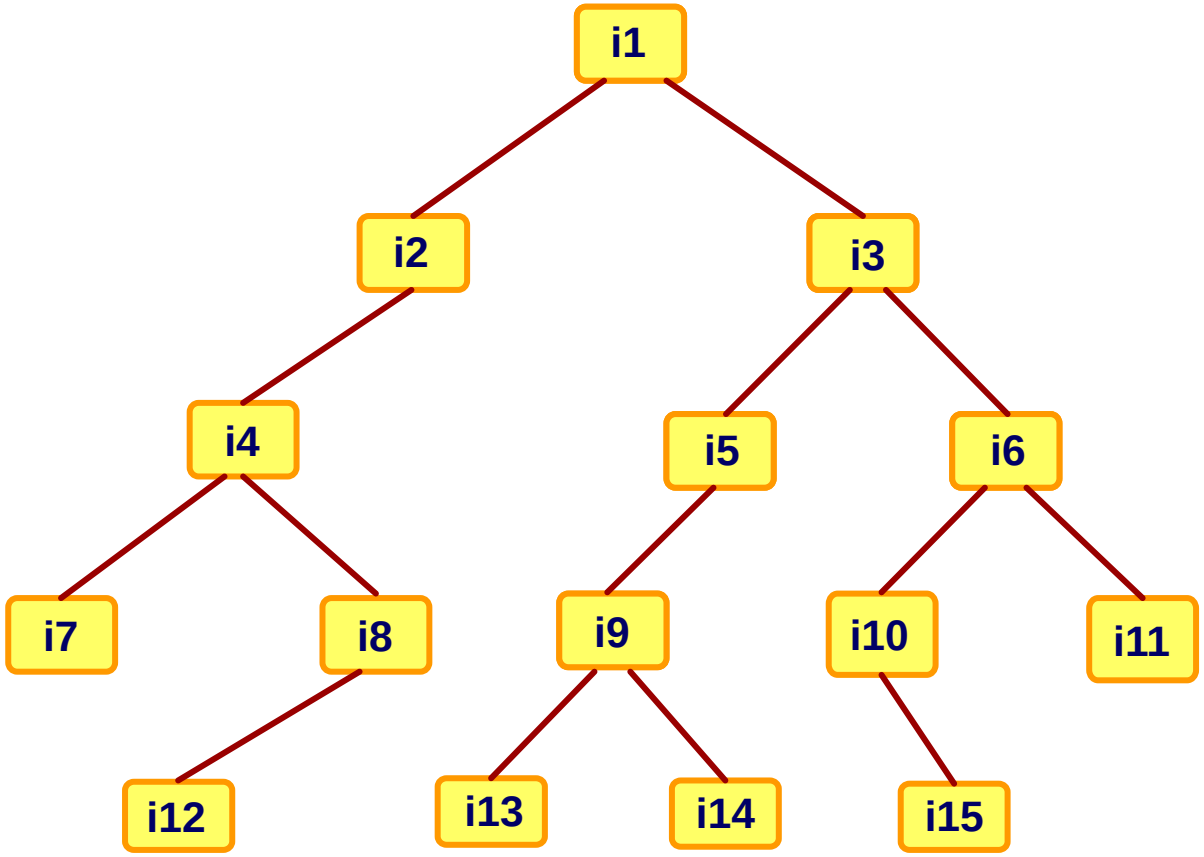
Grado de un ÁRBOL: máximo grado de todos los nodos

Peso de un ÁRBOL: número de nodos terminales o nodos hoja

Altura de un ÁRBOL: número de nodos de la rama más larga



ÁRBOL BINARIO

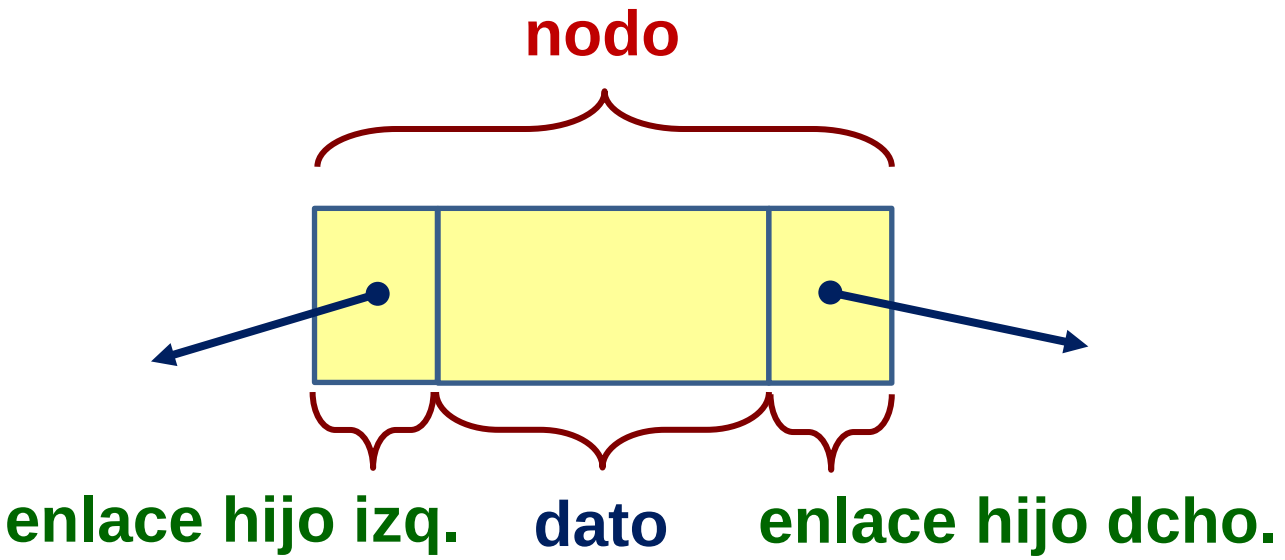


T3. Tipos Abstractos de Datos (TAD)

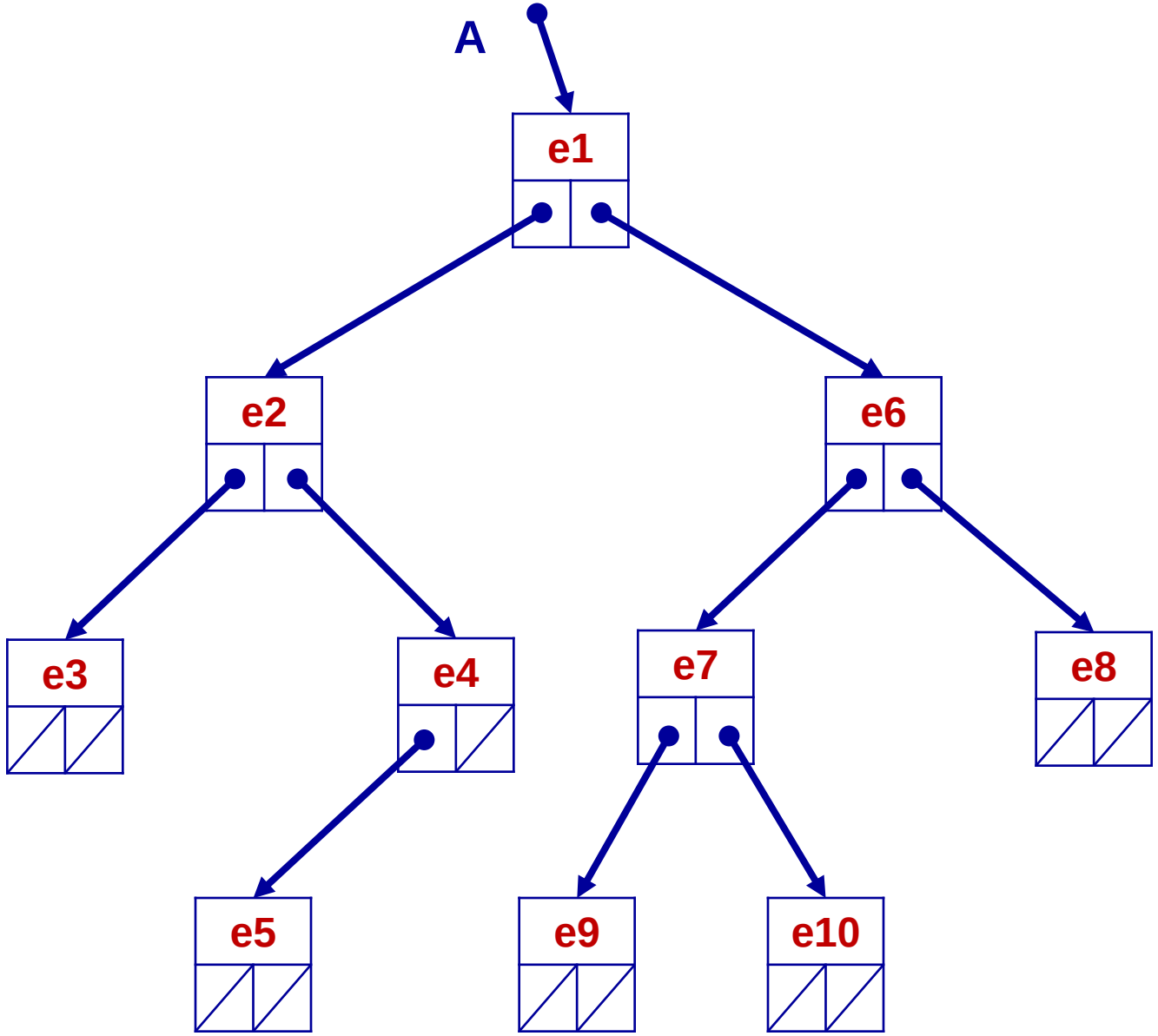
- **Árboles.**
  - Conceptos generales
  - **Realización del TAD Árbol Binario**
  - Recorridos de Árboles Binarios
  - Árboles Binarios de Búsqueda (ABB)
  - Árboles Equilibrados (AVL)
  - Montículos

UNIVERSIDAD NEBRIJA

ÍNDICE

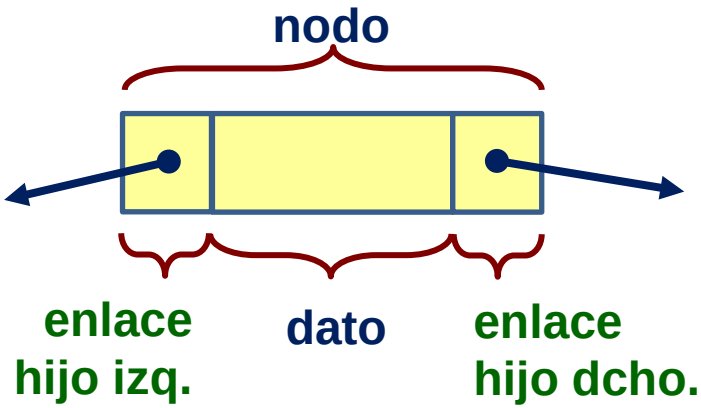

nodo
enlace hijo izq.
dato
enlace hijo dcho.

TAD Árbol Binario



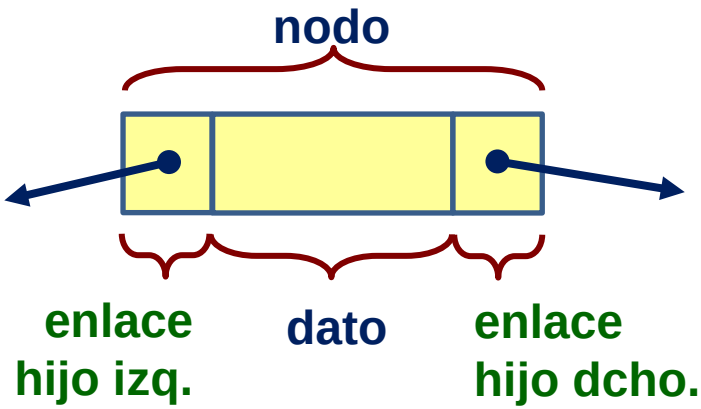
TAD Árbol Binario

```
//---------------Clase CDato
class CDato {   ...};

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

        const CDato &getDato() const;
        void setDato(const CDato &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;
};
```



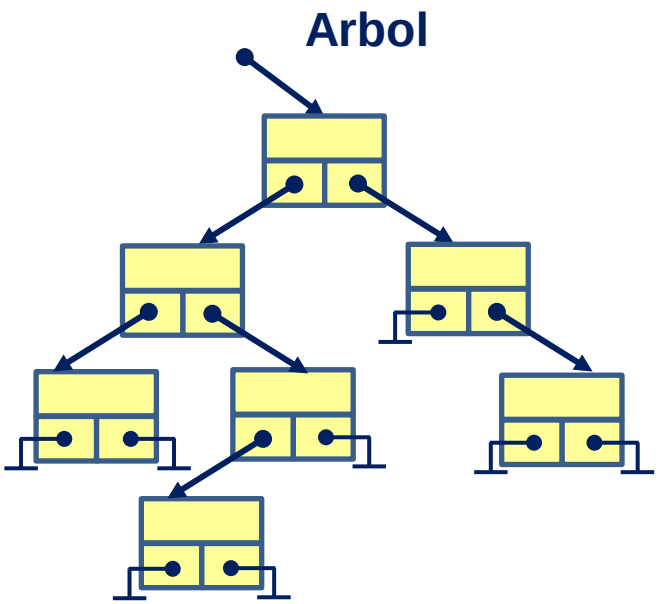
TAD Árbol Binario

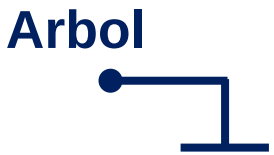
```
//---------------Clase Arbol Binario
class Arbol {
    private:
        shared_ptr<Nodo> raiz;
    public:
        Arbol():raiz(nullptr){};
        Arbol(CDato const &dato);

        bool empty() const;

        void addHizq(Arbol const &Ai);
        void addHdch(Arbol const &Ad);

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

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

};
```



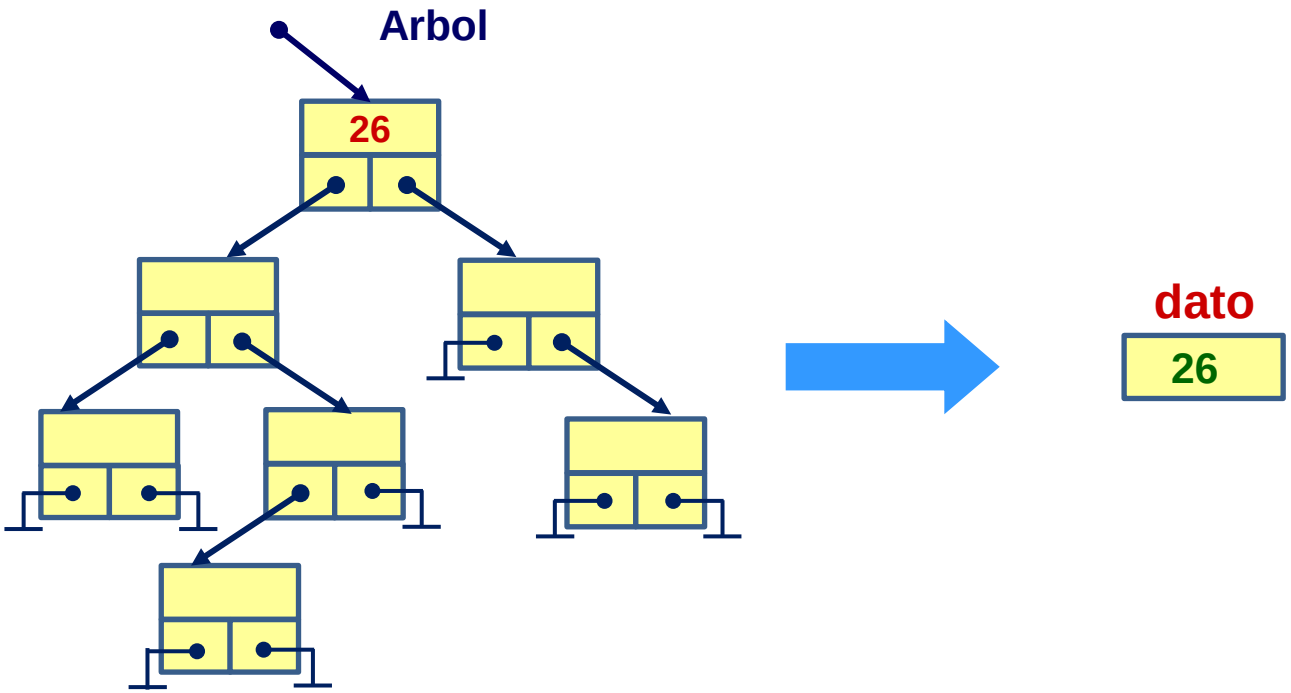
TAD Árbol Binario

```
bool Arbol::empty() const {
    return (raiz == nullptr);
}
```

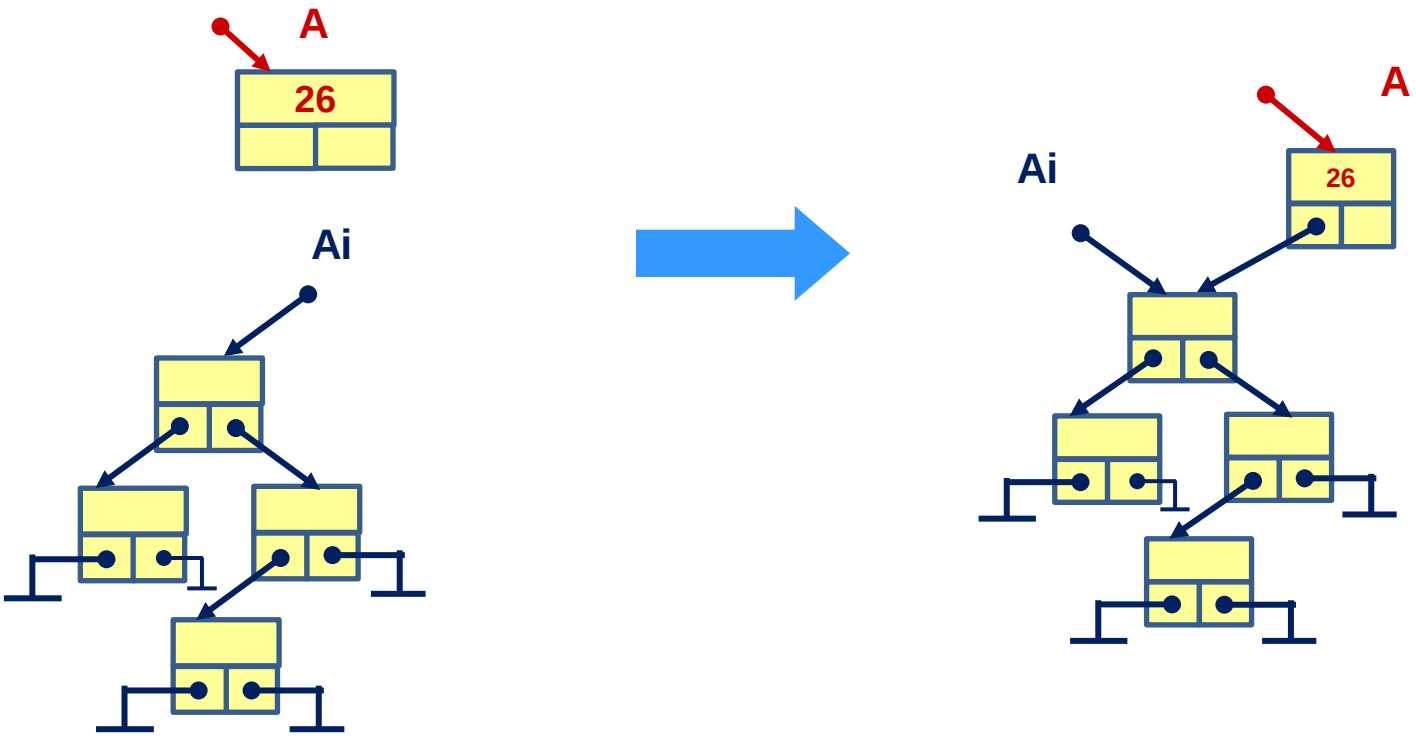




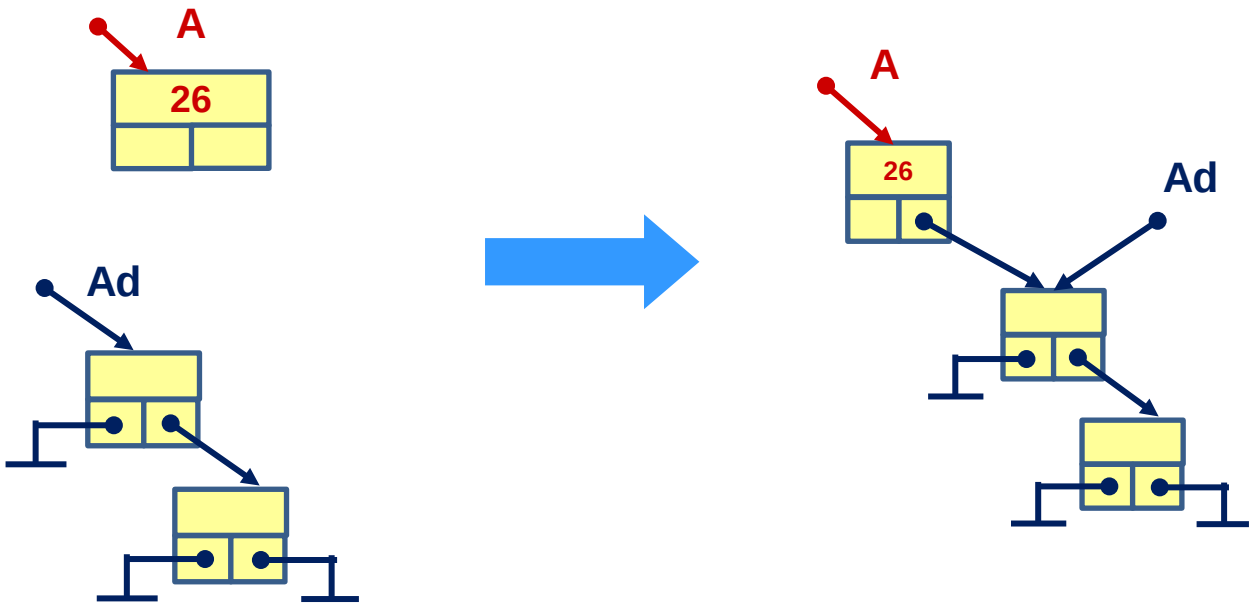
```
CDato const &Arbol::getDatoNodo () const {
    return raiz->getDato();
}
```



```
void Arbol::addHizq(Arbol const &Ai) {
    raiz->setHizq(Ai.raiz);
}
```

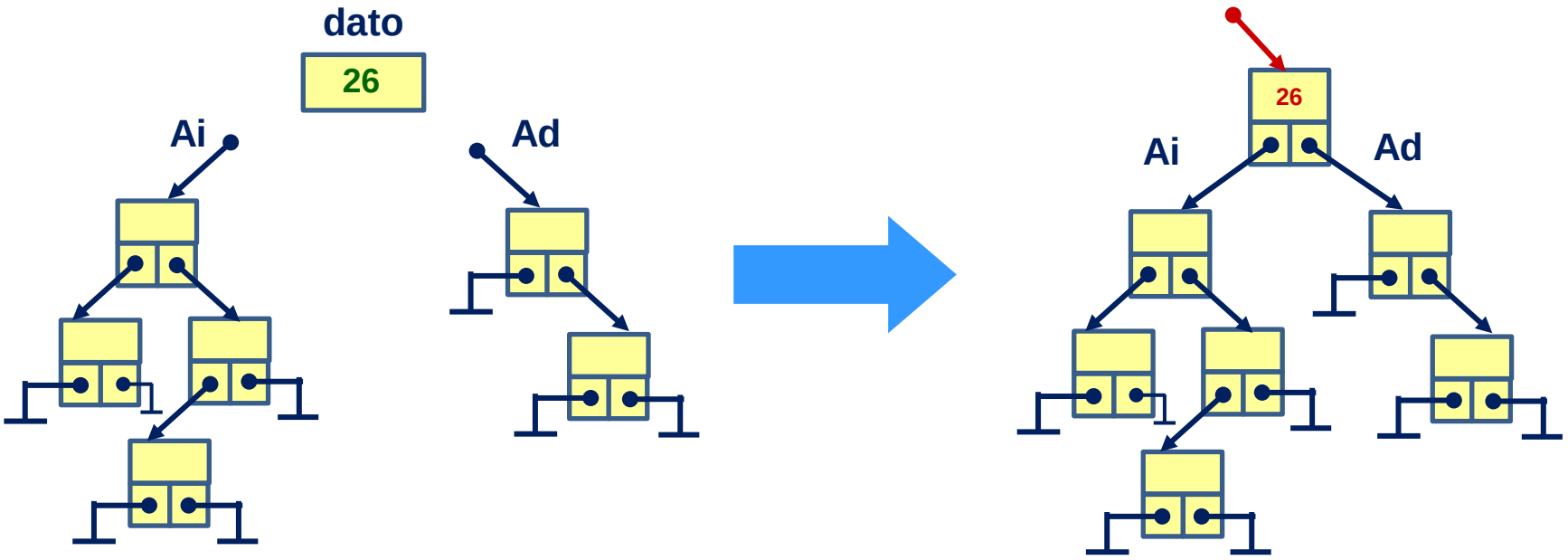


```
void Arbol::addHdch(Arbol const &Ad) {
    raiz->setHdch(Ad.raiz);
}
```

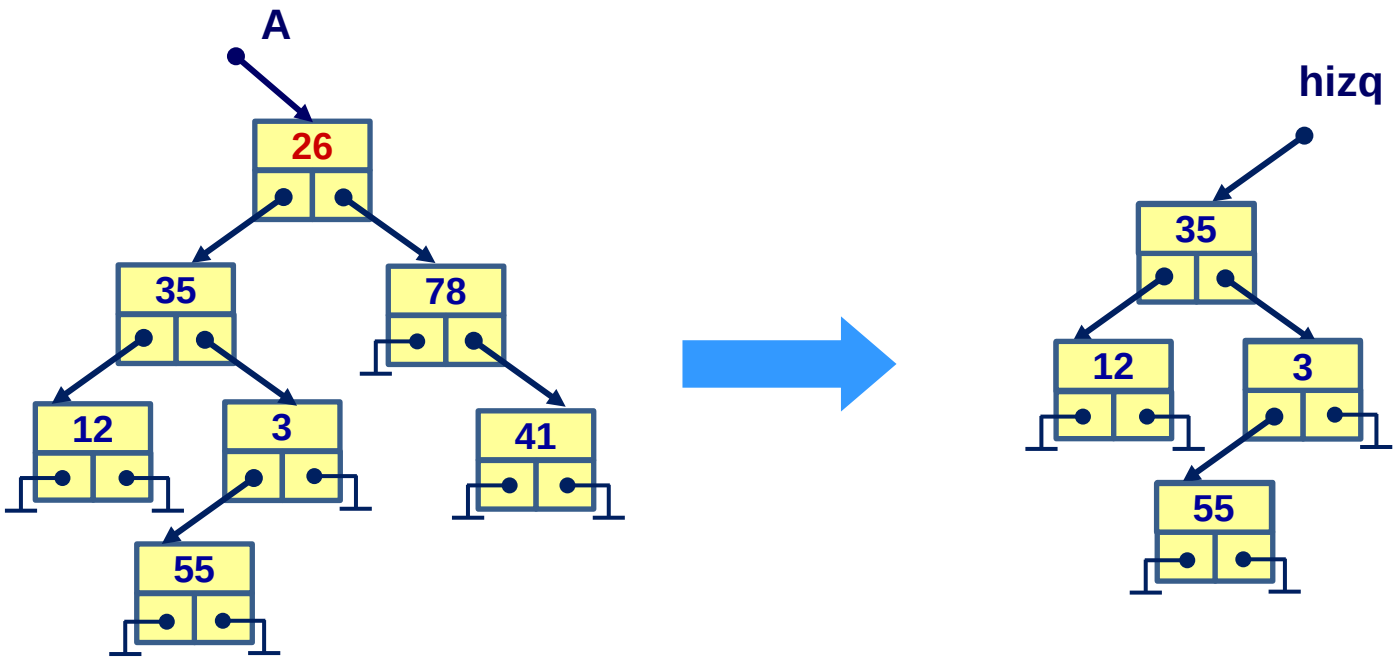


TAD Árbol Binario

```
void Arbol::construirArbol (Arbol const &Ai, Arbol const &Ad, CDato const &dato) {
    raiz = make_shared<Nodo>(Nodo{dato});
    raiz->setHizq(Ai.raiz);
    raiz->setHdch(Ad.raiz);
}
```

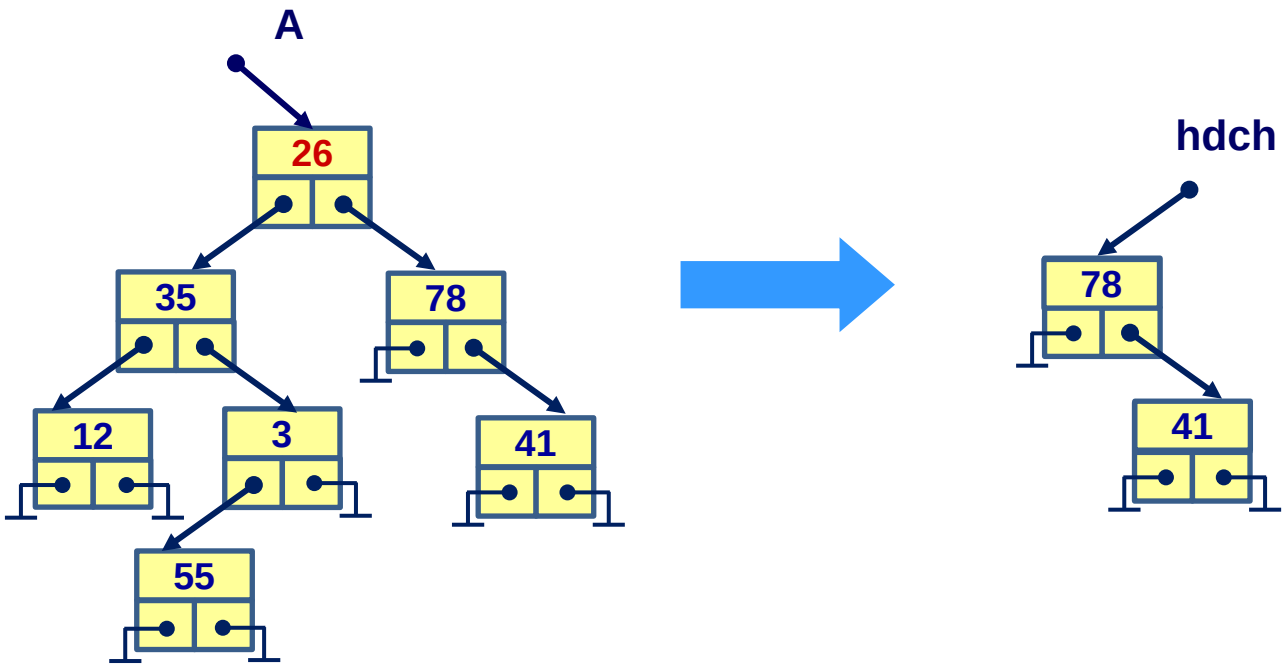


```
const shared_ptr<Nodo> &Nodo::getHizq() const {
    return hizq;
}

const shared_ptr<Nodo> &Arbol::getHiNodo() const {
    return raiz->getHizq();
}
```



TAD Árbol Binario

```
const shared_ptr<Nodo> &Nodo::getHdch() const {
    return hdch;
}

const shared_ptr<Nodo> &Arbol::getHdNodo() const {
    return raiz->getHdch();
}
```



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

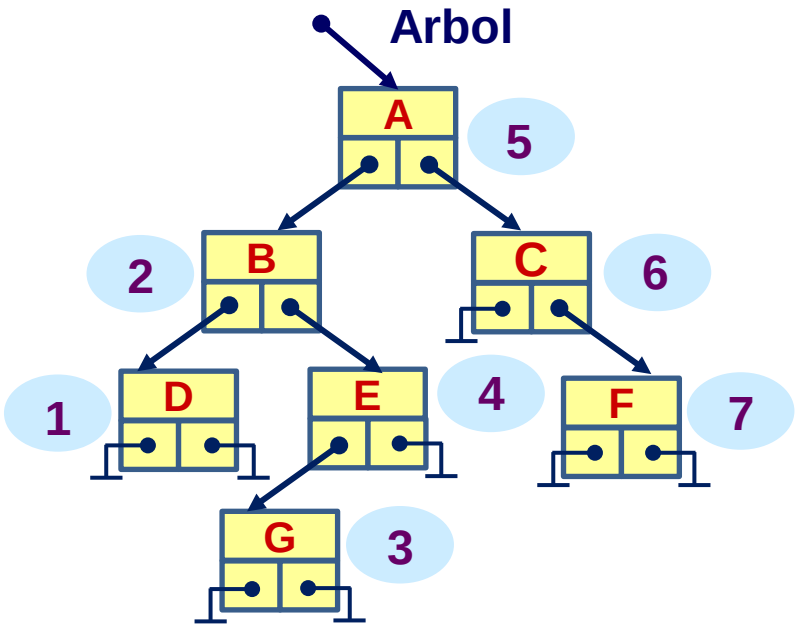
UNIVERSIDAD NEBRIJA

ÍNDICE

El recorrido en **orden simétrico** u **orden central** o **inorden** procesa los nodos de un árbol según la versión **IRD** o normal de expresiones.

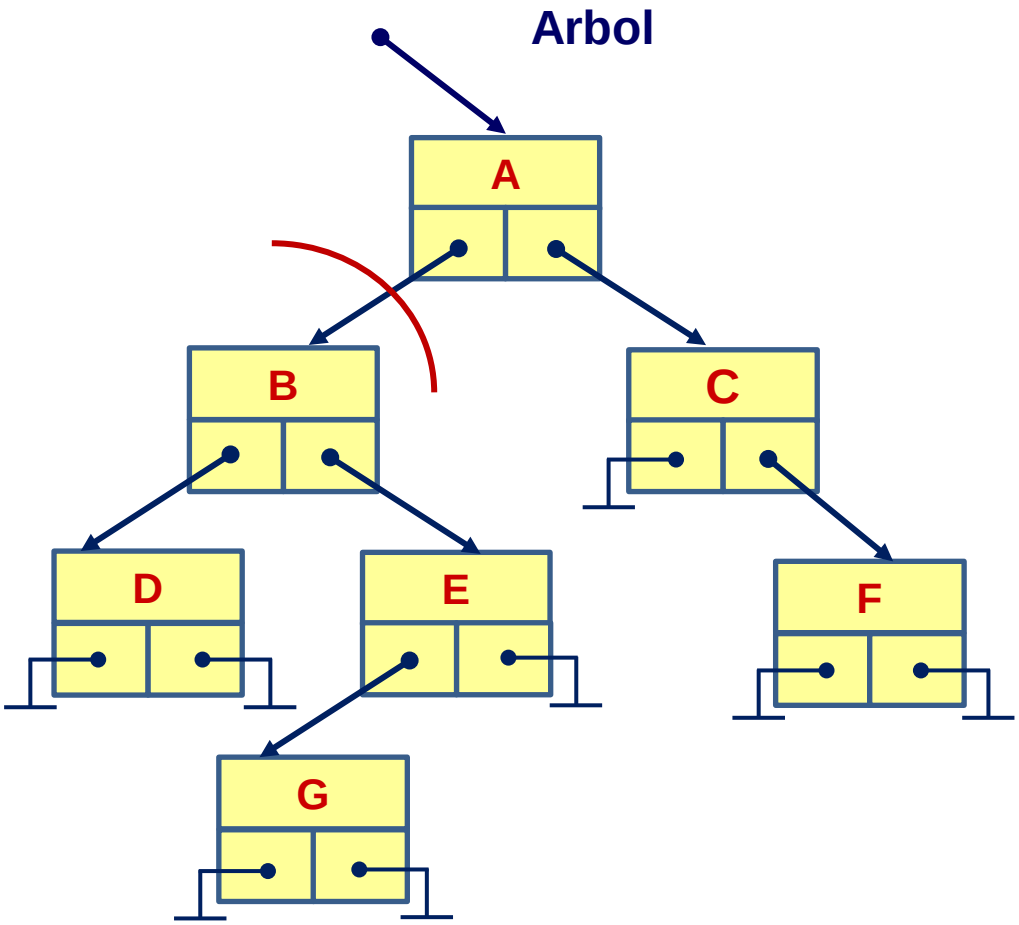
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



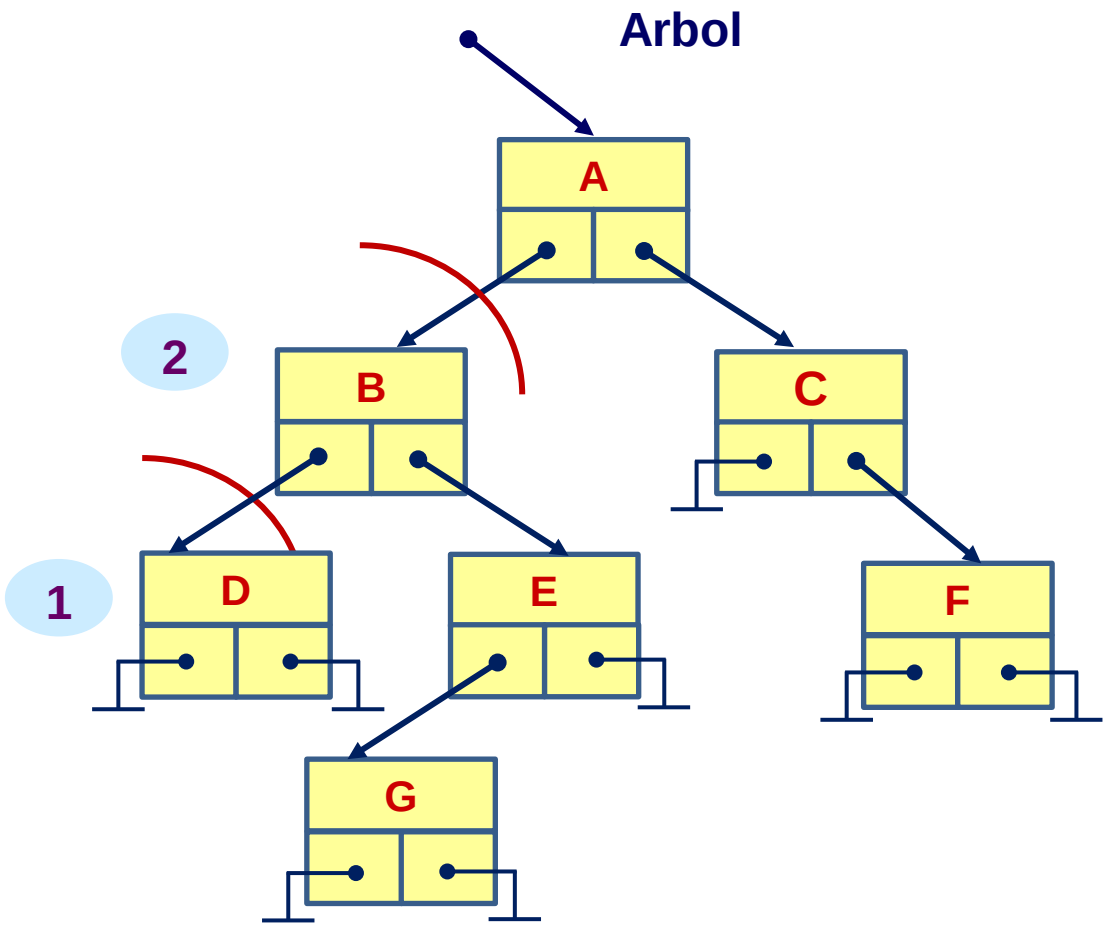
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



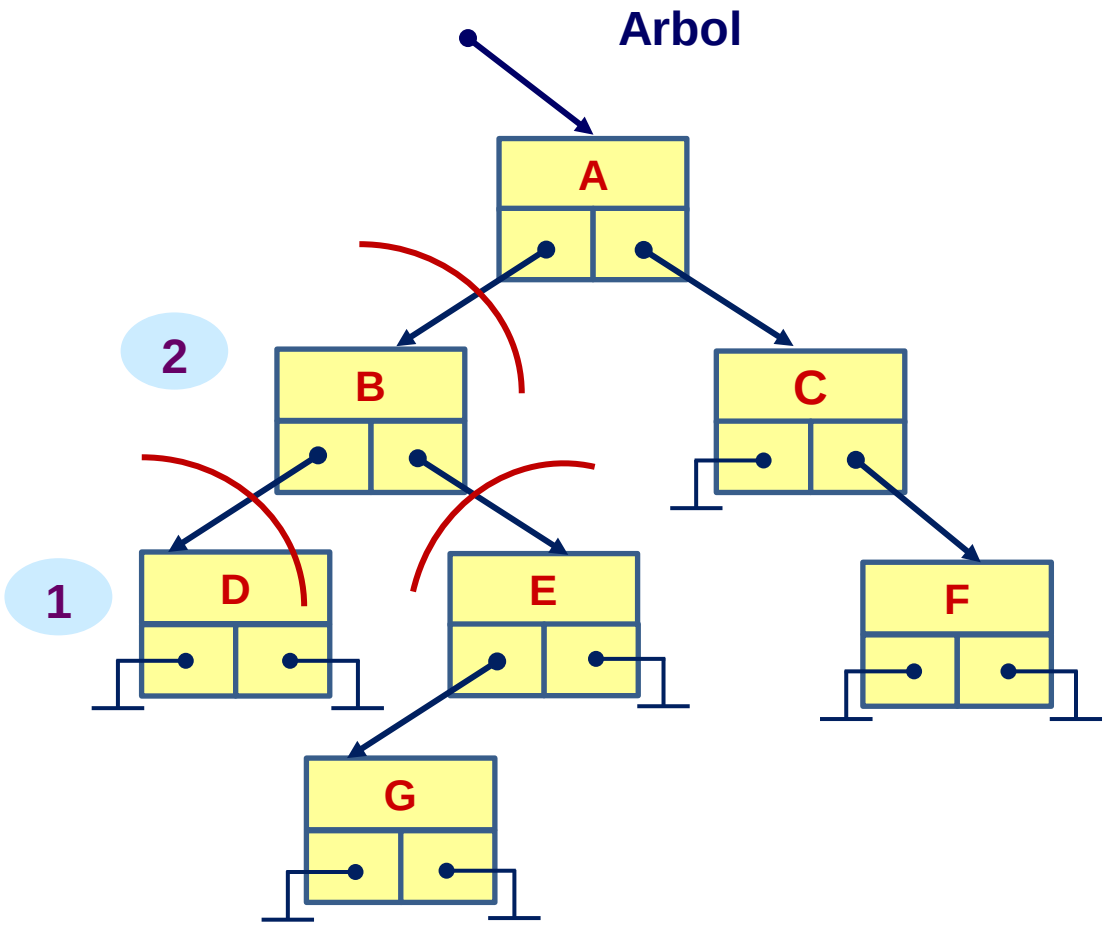
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



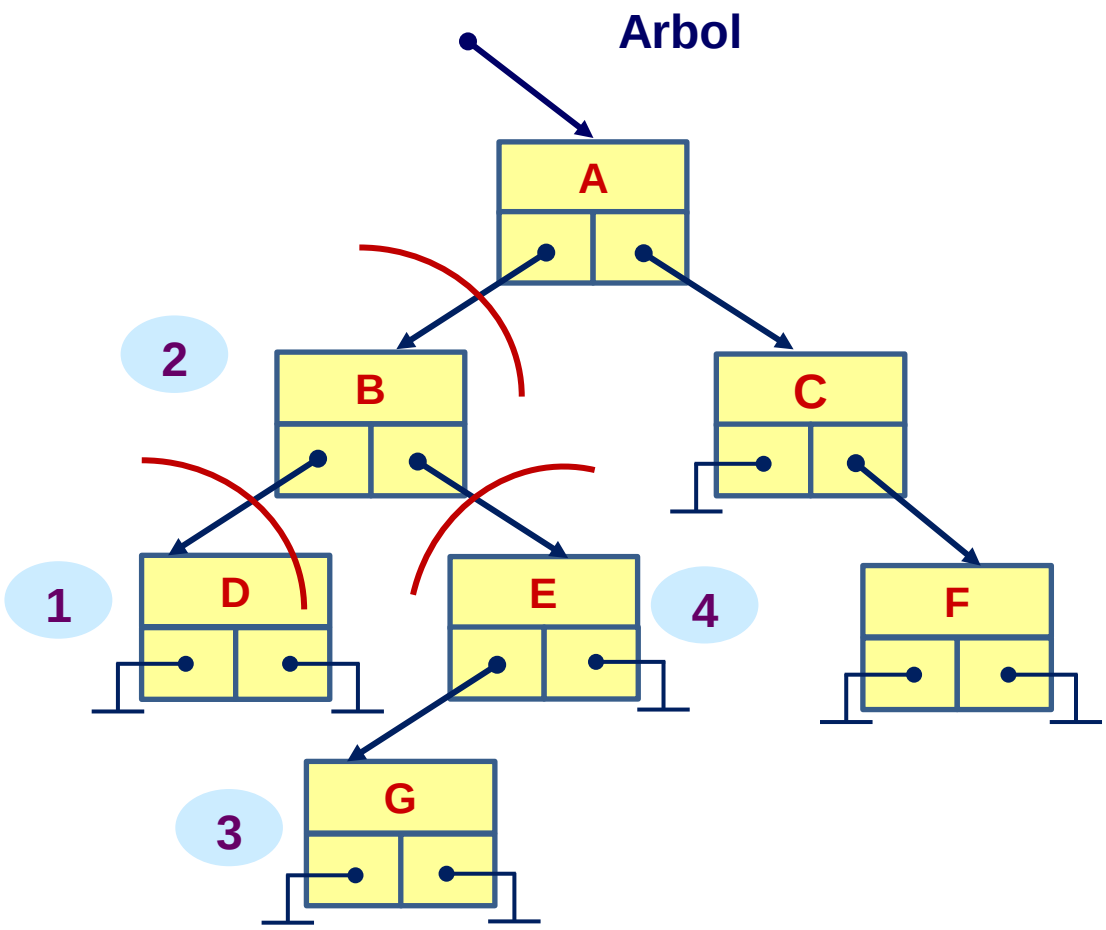
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



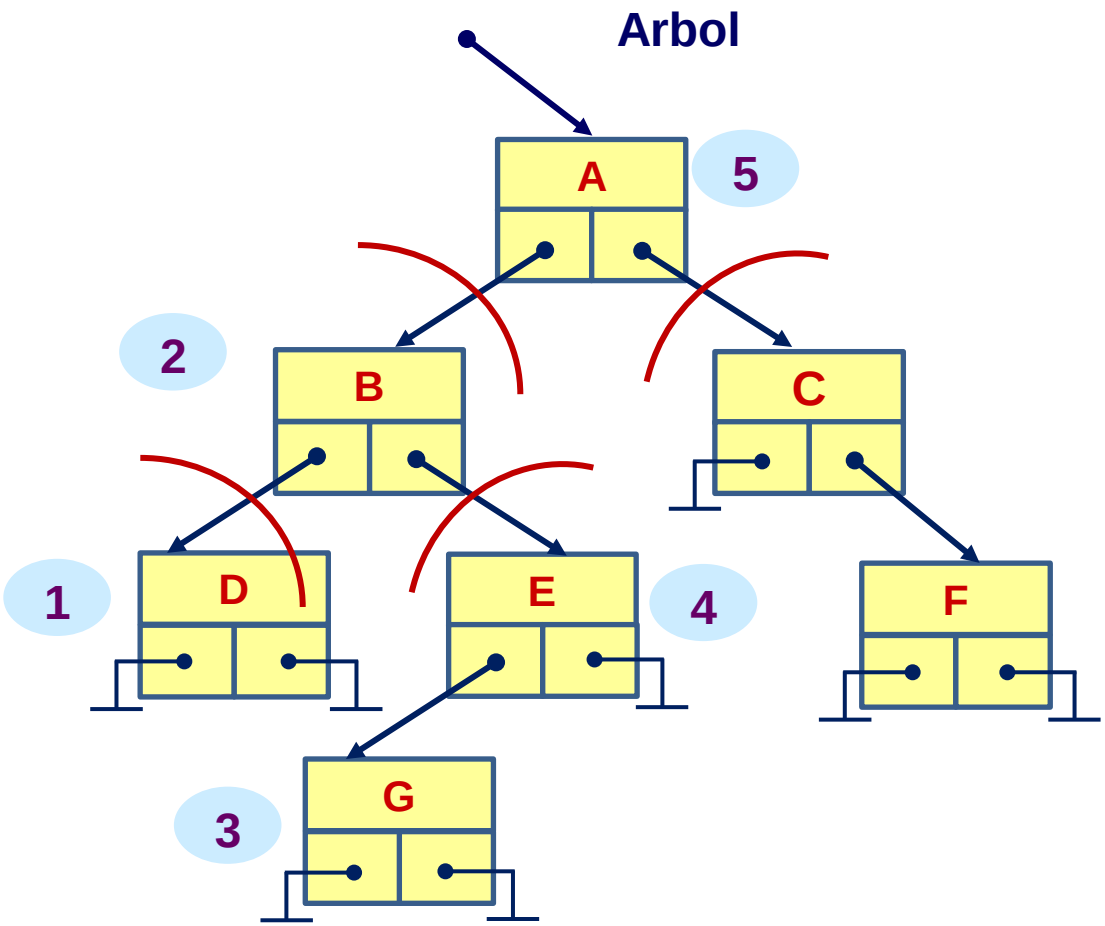
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



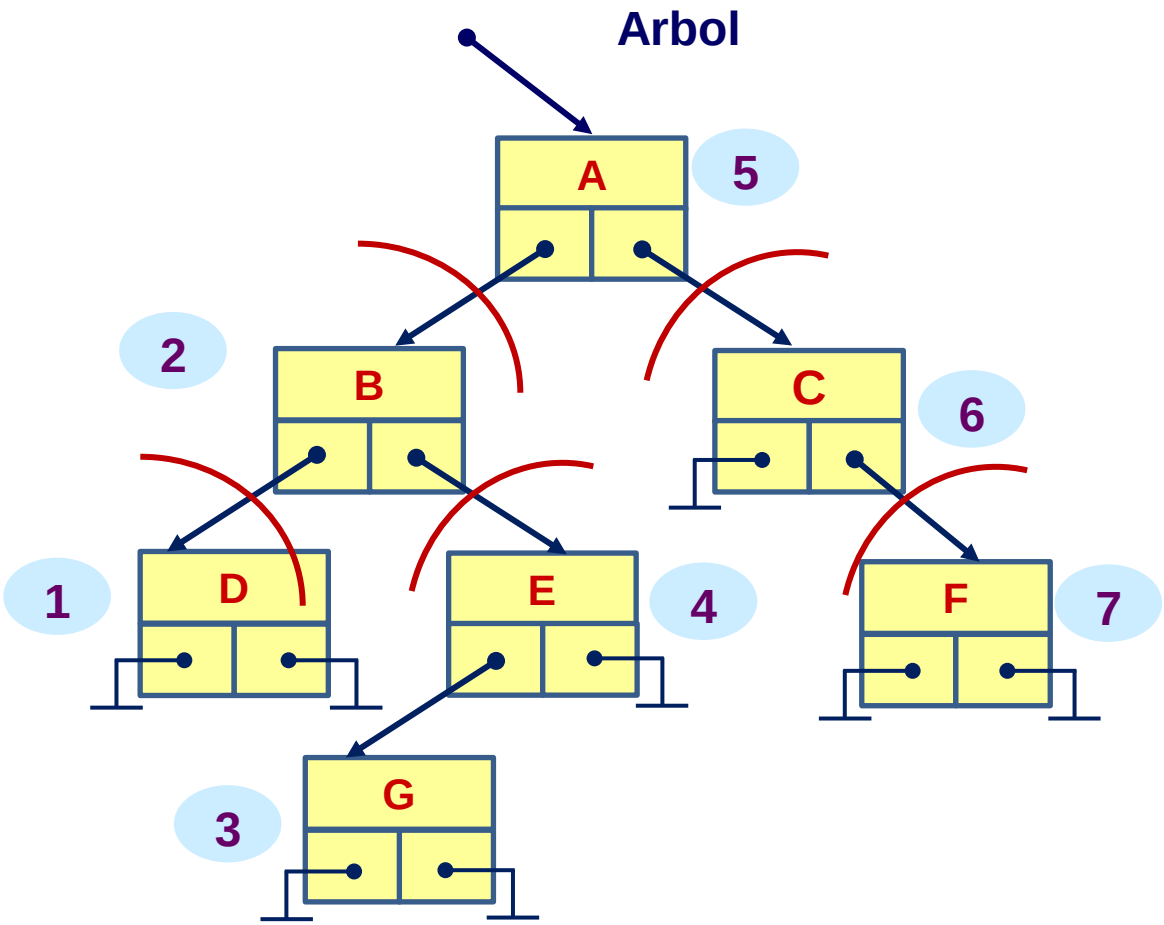
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



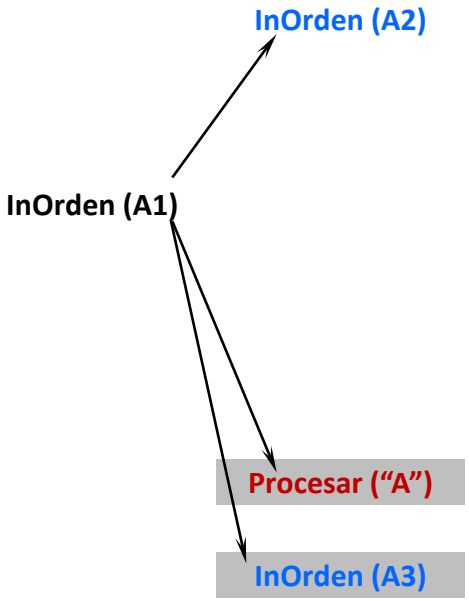
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



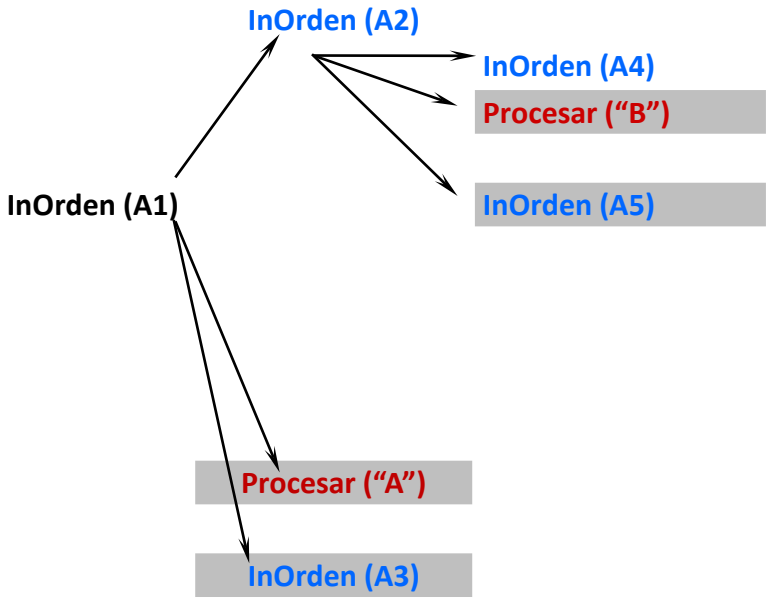
Llamadas recursivas

Nodos recorridos

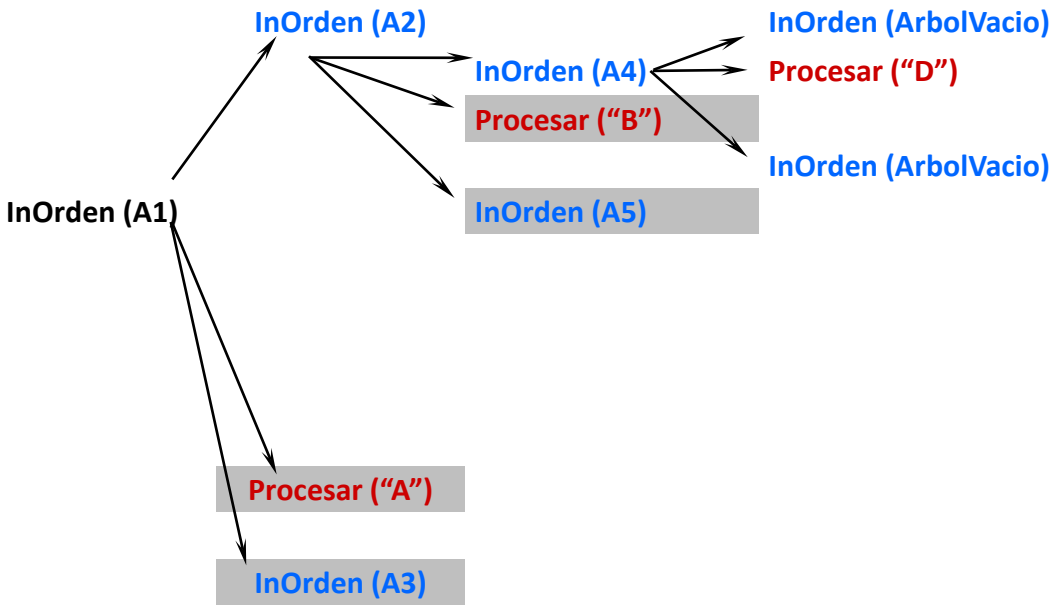


Llamadas recursivas

Nodos recorridos



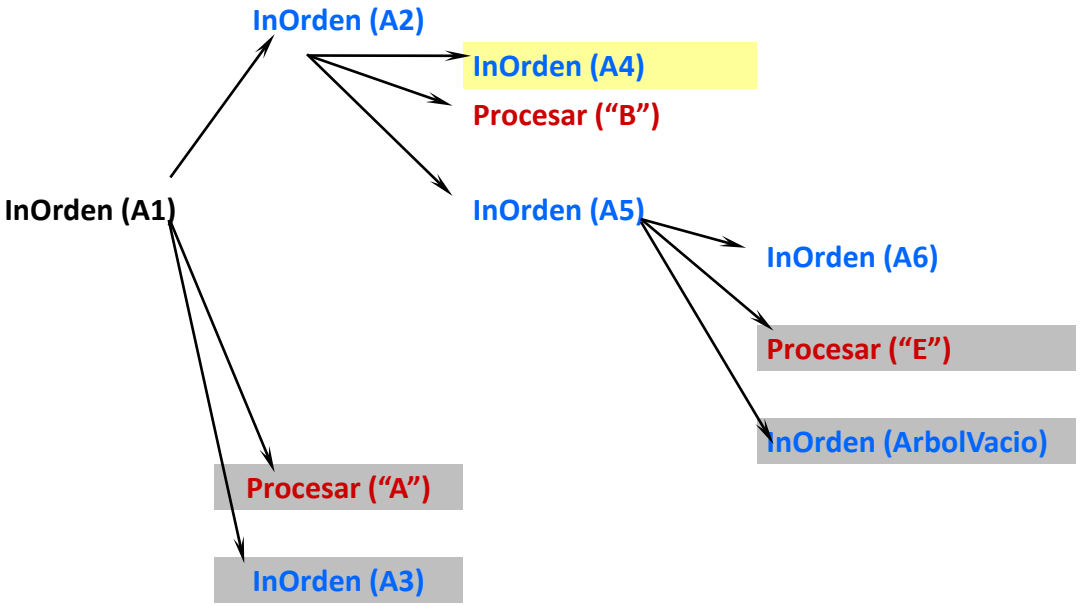
Llamadas recursivas



Nodos recorridos

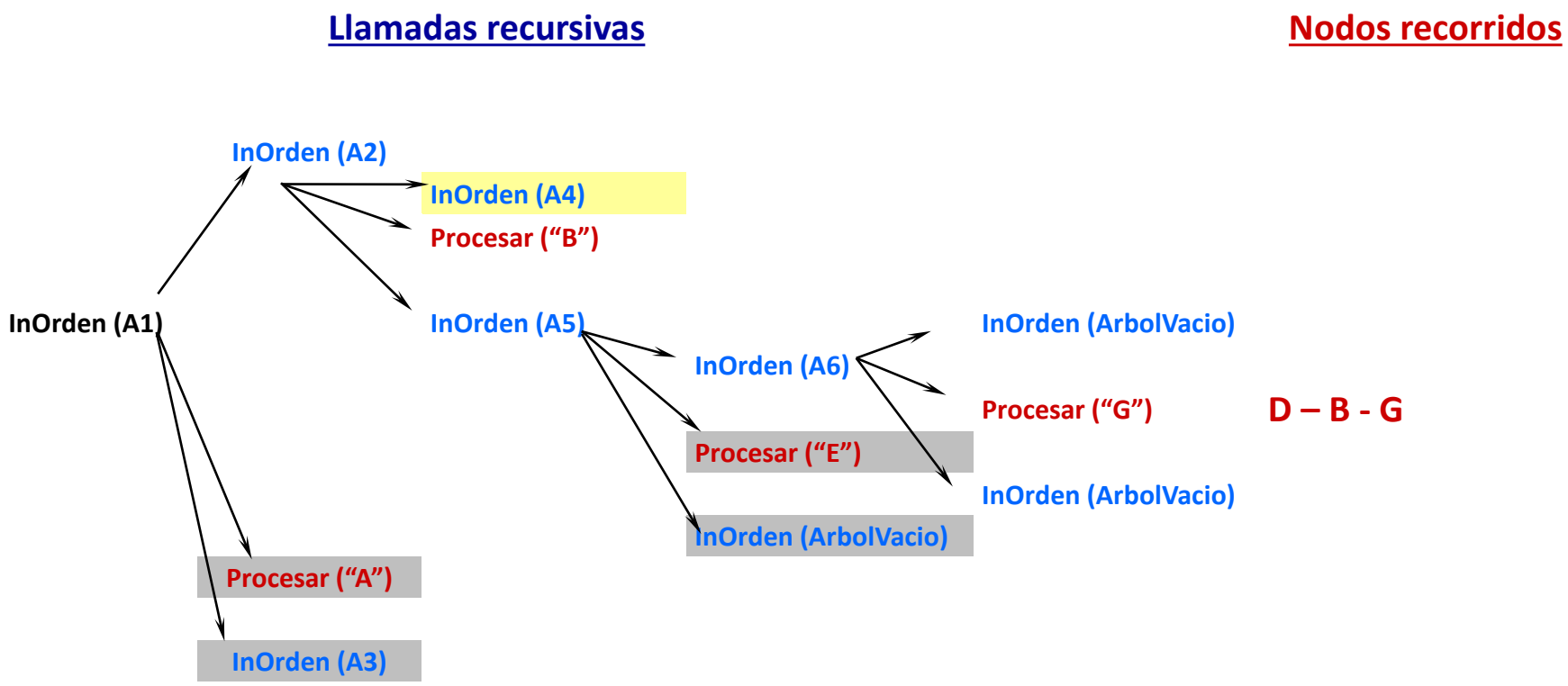
D

Llamadas recursivas

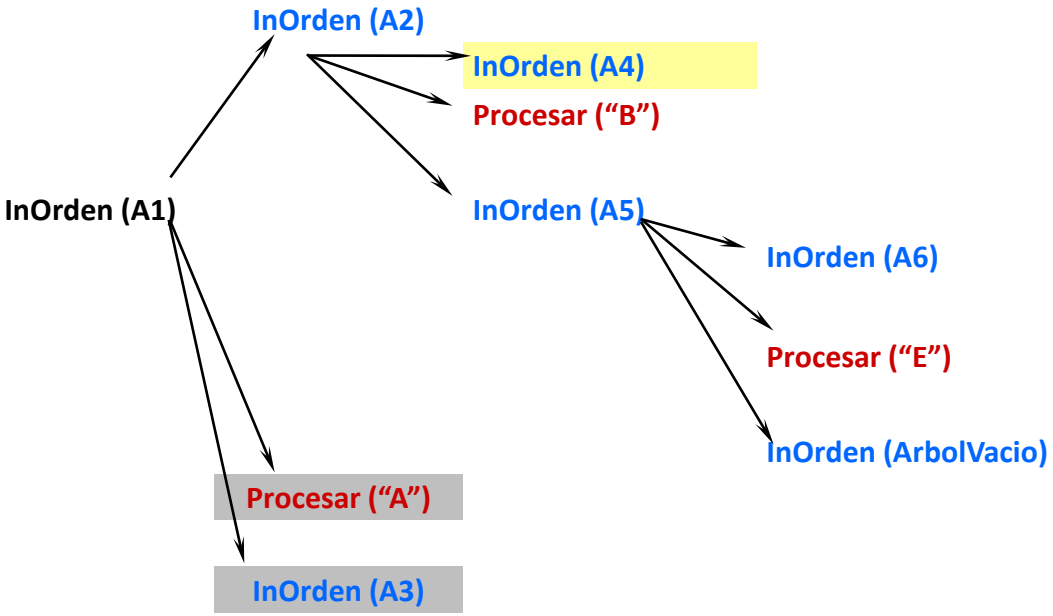


Nodos recorridos

D - B


Llamadas recursivas
Nodos recorridos
InOrden (A2)
InOrden (A4)
Procesar ("B")
InOrden (A1)
InOrden (A5)
InOrden (ArbolVacio)
InOrden (A6)
Procesar ("G")
D – B - G
Procesar ("E")
InOrden (ArbolVacio)
InOrden (ArbolVacio)
Procesar ("A")
InOrden (A3)

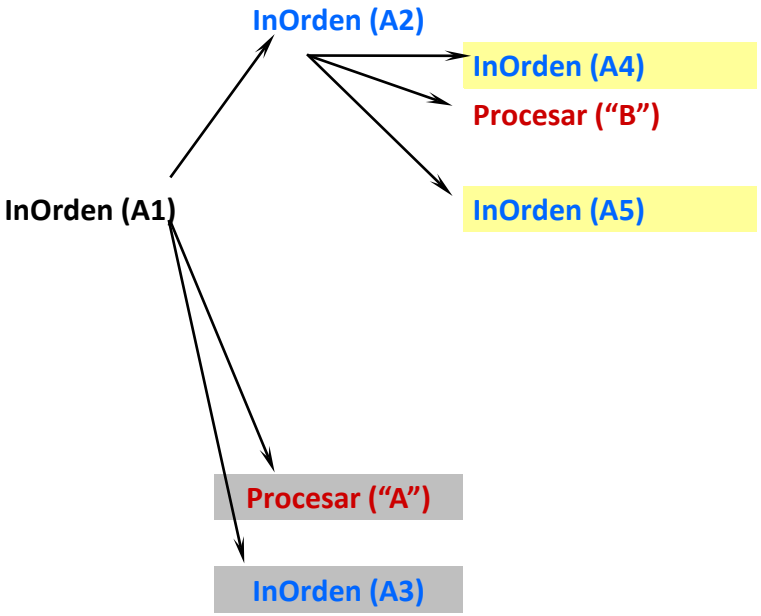
Llamadas recursivas



Nodos recorridos

D – B – G – E

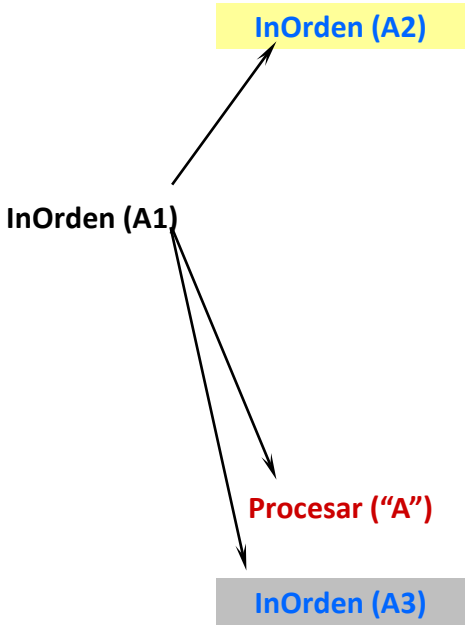
Llamadas recursivas



Nodos recorridos

D – B – G – E

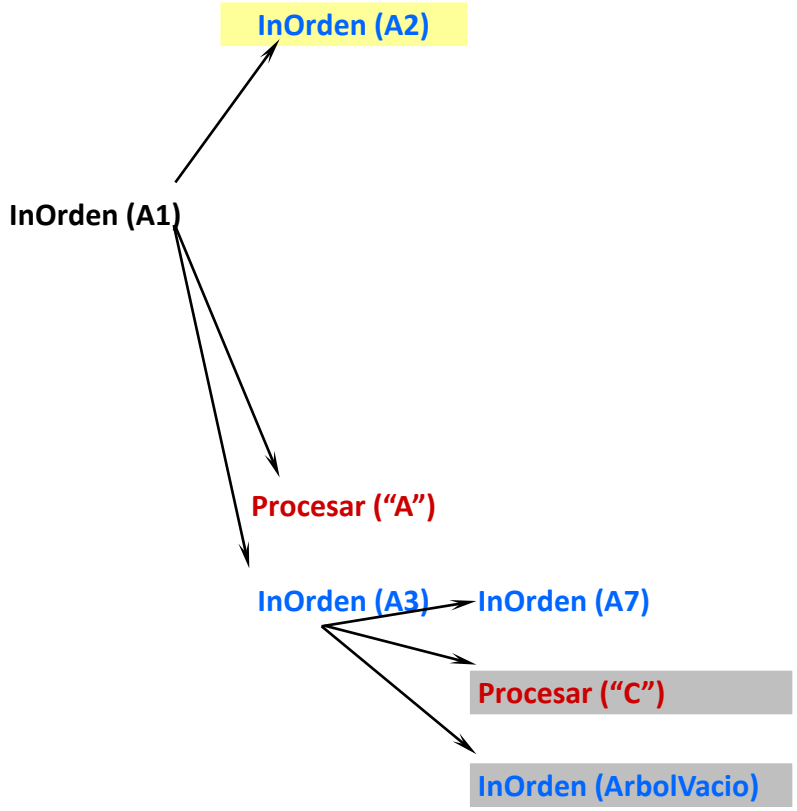
Llamadas recursivas



Nodos recorridos

D – B – G – E – A

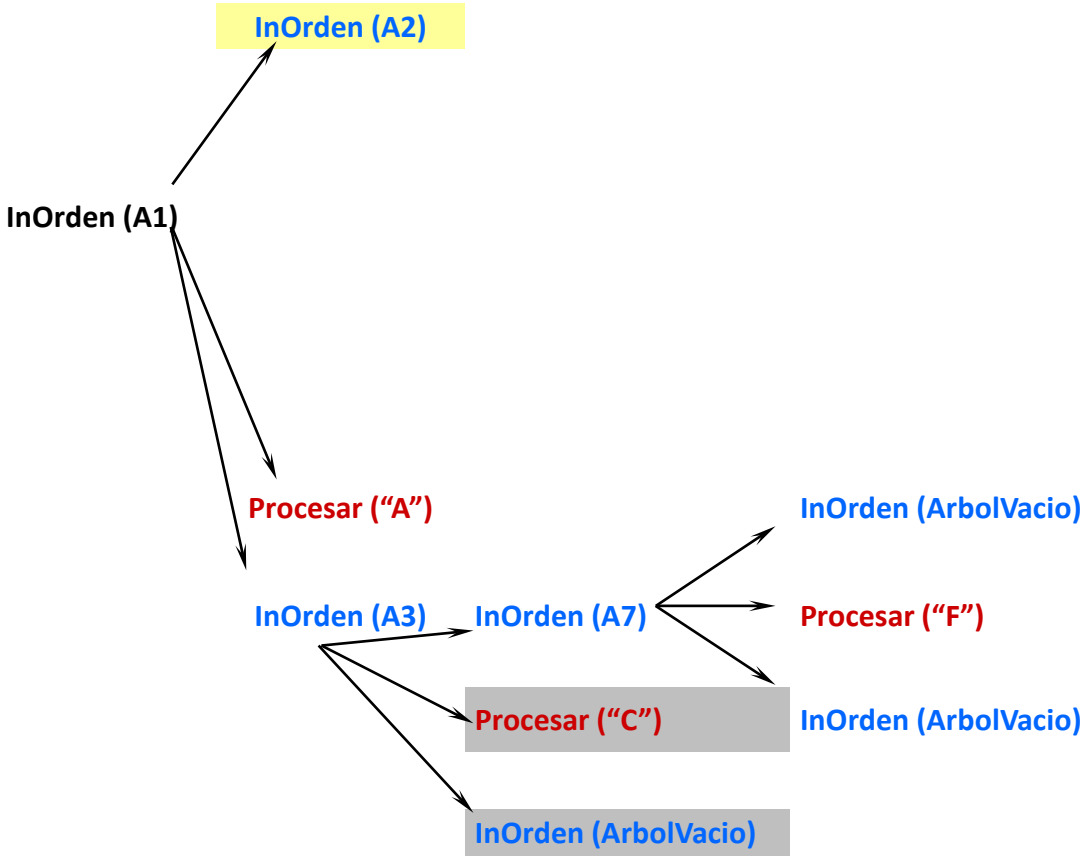
Llamadas recursivas



Nodos recorridos

D – B – G – E – A

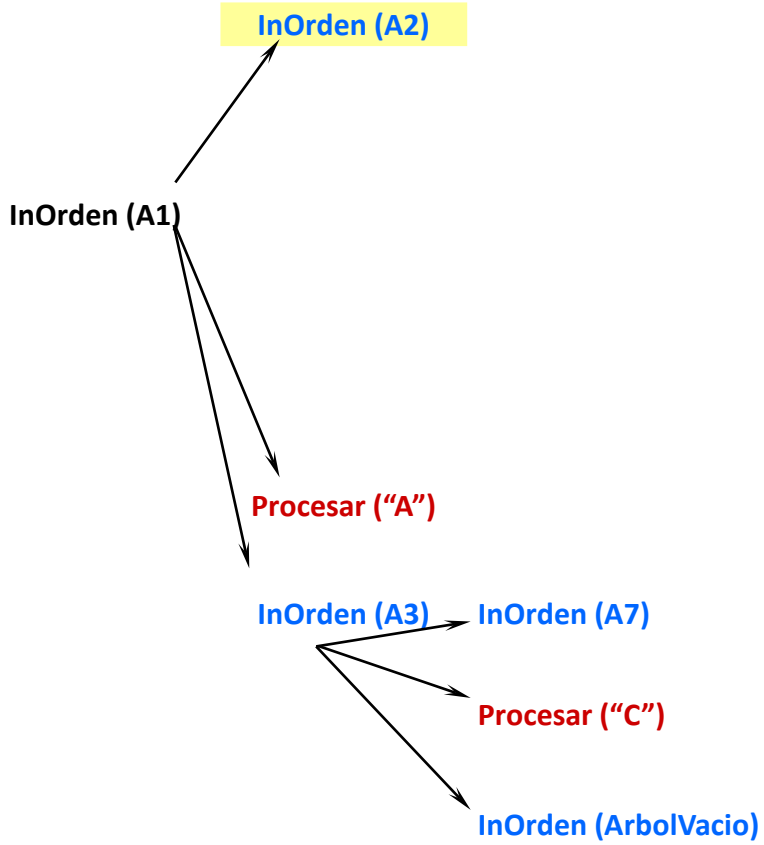
Llamadas recursivas



Nodos recorridos

D – B – G – E – A – F

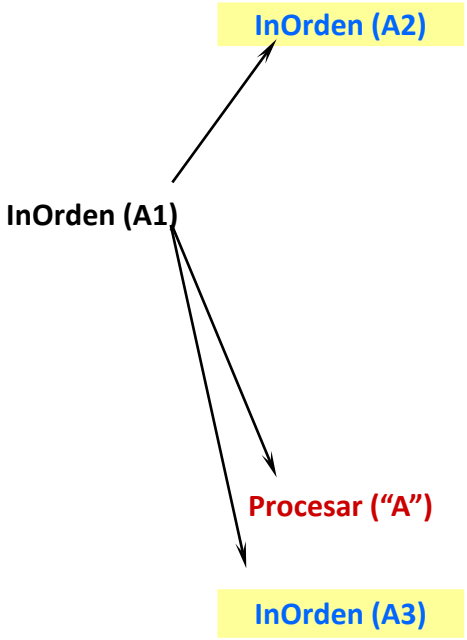
Llamadas recursivas



Nodos recorridos

D – B – G – E – A – F – C

Llamadas recursivas



Nodos recorridos

D – B – G – E – A – F – C

ÁRBOLES

Llamadas recursivas

Nodos recorridos

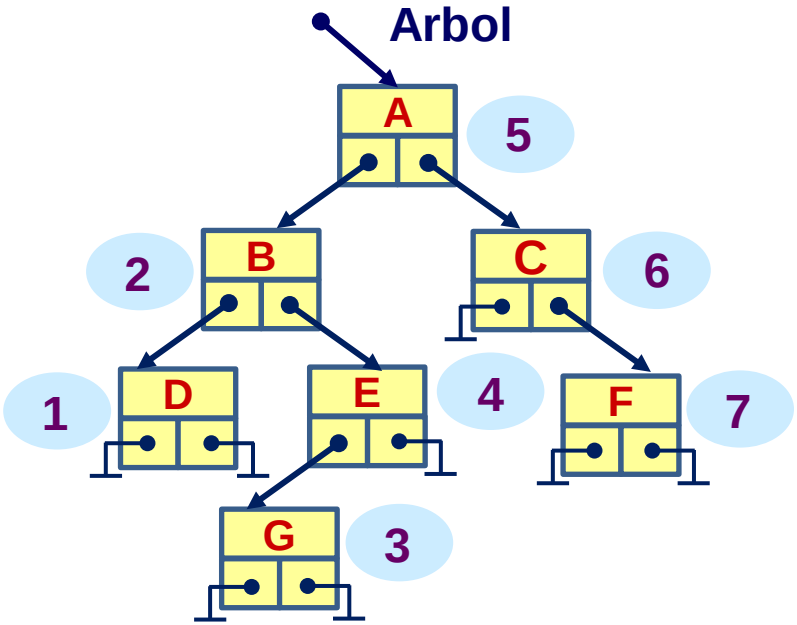
InOrden (A1)

D – B – G – E – A – F – C

El recorrido en **orden simétrico** u **orden central** o **inorden** procesa los nodos de un árbol según la versión **IRD** o normal de expresiones.

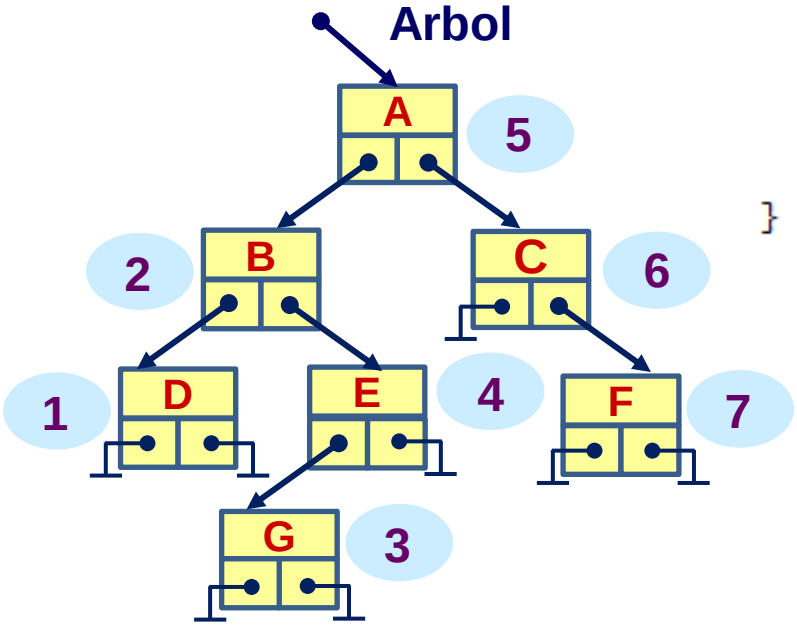
El recorrido se hace en el siguiente orden:

- **Recorrer el subárbol izquierdo (I)**
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol derecho (D)**



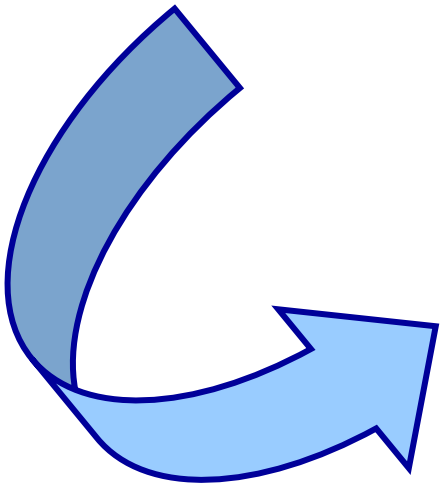
D, B, G, E, A, C, F

D, B, G, E, A, C, F




```
void Arbol::InOrden(shared_ptr<Nodo> A) const {
    shared_ptr<Nodo> Ai, Ad;
    CDato dato;

    if (A != nullptr) {
        Ai = A->getHizq();
        Ad = A->getHdch();
        dato = A->getDato();

        InOrden (Ai);           //  I
        A->procesarNodo();      //  R
        InOrden (Ad);           //  D
    }
}

void Arbol::recorrerInOrden() const {
    if (!empty()) {
        InOrden (raiz);
    }
}
```

```
void Arbol::InOrden(shared_ptr<Nodo> A) const {
    shared_ptr<Nodo> Ai, Ad;
    CDato dato;

    if (A != nullptr) {
        Ai = A->getHizq();
        Ad = A->getHdch();
        dato = A->getDato();

        InOrden (Ai);          //  I
        A->procesarNodo();     //  R
        InOrden (Ad);          //  D
    }
}
```

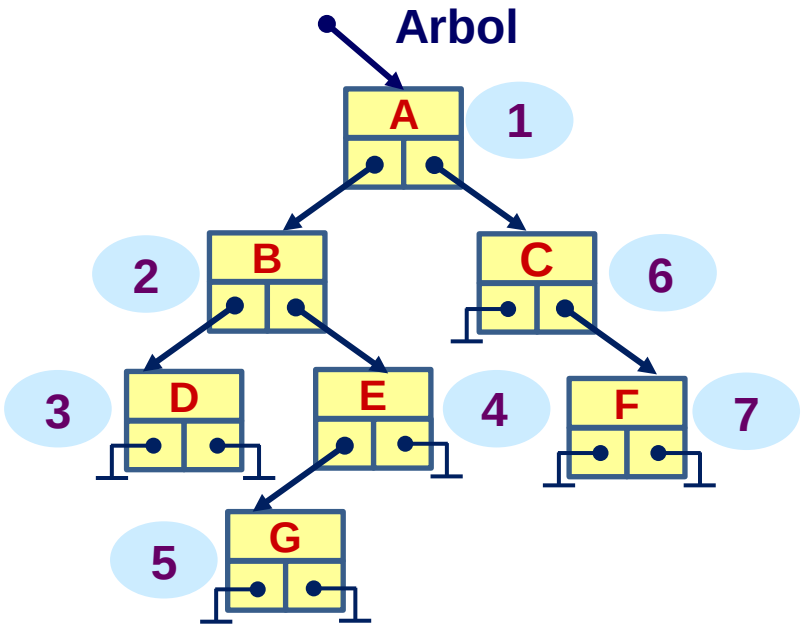


```
void Arbol::InOrden(shared_ptr<Nodo> A) const {
    if (A != nullptr) {
        InOrden (A->getHizq());          // I
        A->procesarNodo();               // R
        InOrden (A->getHdch());          // D
    }
}
```

El recorrido **preorden, notación prefija** o **notación polaca** (primero aparece el **operador** y después los **dos operandos**) procesa los nodos de un árbol según la versión **RID**.

El recorrido se hace en el siguiente orden:

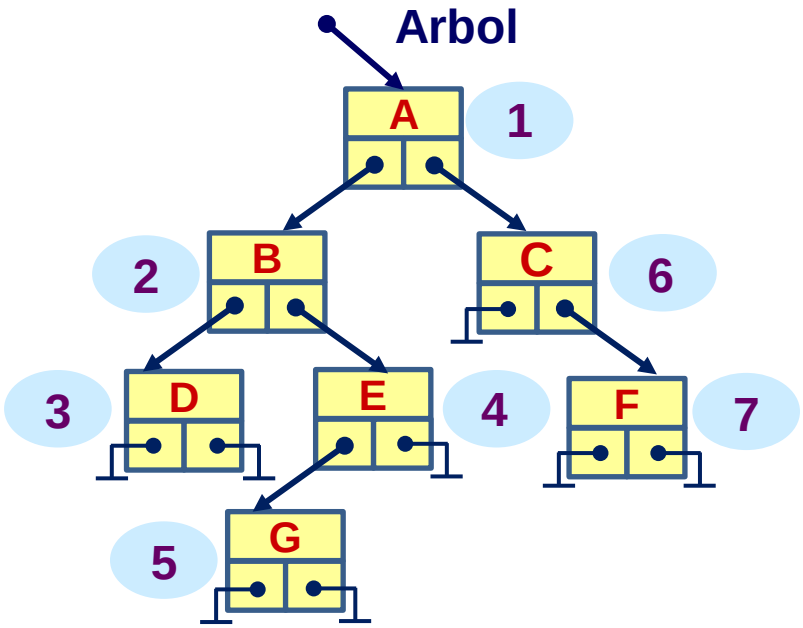
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol izquierdo (I)**
- **Recorrer el subárbol derecho (D)**



El recorrido **preorden, notación prefija** o **notación polaca** (primero aparece el **operador** y después los **dos operandos**) procesa los nodos de un árbol según la versión **RID**.

El recorrido se hace en el siguiente orden:

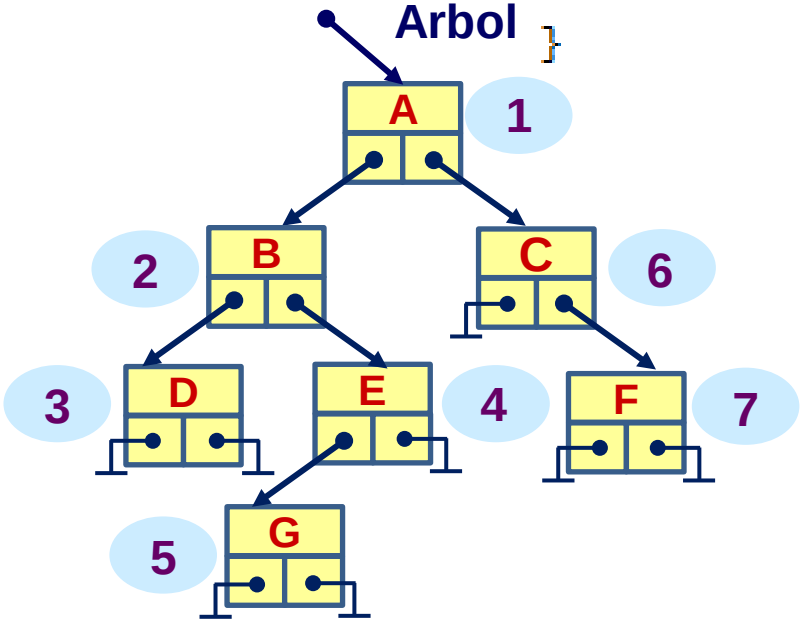
- **Procesar el nodo raíz (R)**
- **Recorrer el subárbol izquierdo (I)**
- **Recorrer el subárbol derecho (D)**



A, B, D, E, G, C, F

ÁRBOLES

```
void Arbol::PreOrden(shared_ptr<Nodo> A) const {
    if (A != nullptr) {
        A->procesarNodo();              // R
        PreOrden (A->getHizq());        // I
        PreOrden (A->getHdch());        // D
    }
}

void Arbol::recorrerPreOrden() const {
    if (!empty()) {
        PreOrden (raiz);
    }
}
```

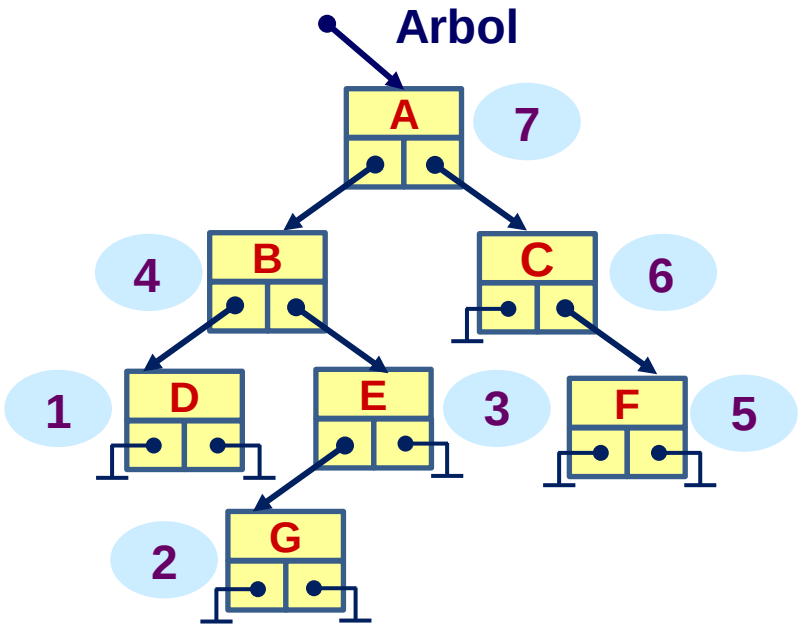


A, B, D, E, G, C, F

El recorrido **postorden, notación postfija de expresiones** o **notación polaca inversa** (primero aparecen los **dos operandos** y después el **operador**) procesa los nodos de un árbol según la versión **IDR**.

El recorrido se hace en el siguiente orden:

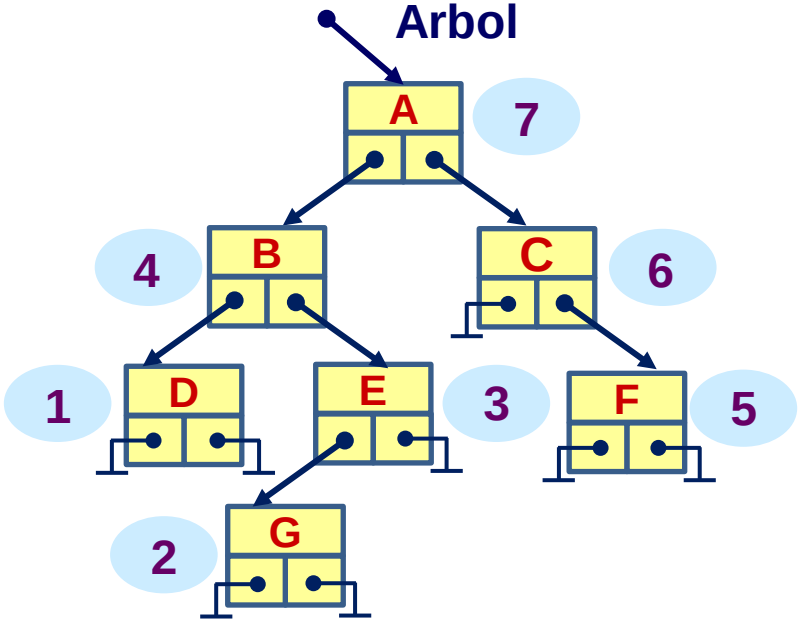
- **Recorrer el subárbol izquierdo (I)**
- **Recorrer el subárbol derecho (D)**
- **Procesar el nodo raíz (R)**



El recorrido **postorden, notación postfija de expresiones** o **notación polaca inversa** (primero aparecen los **dos operandos** y después el **operador**) procesa los nodos de un árbol según la versión **IDR**.

El recorrido se hace en el siguiente orden:

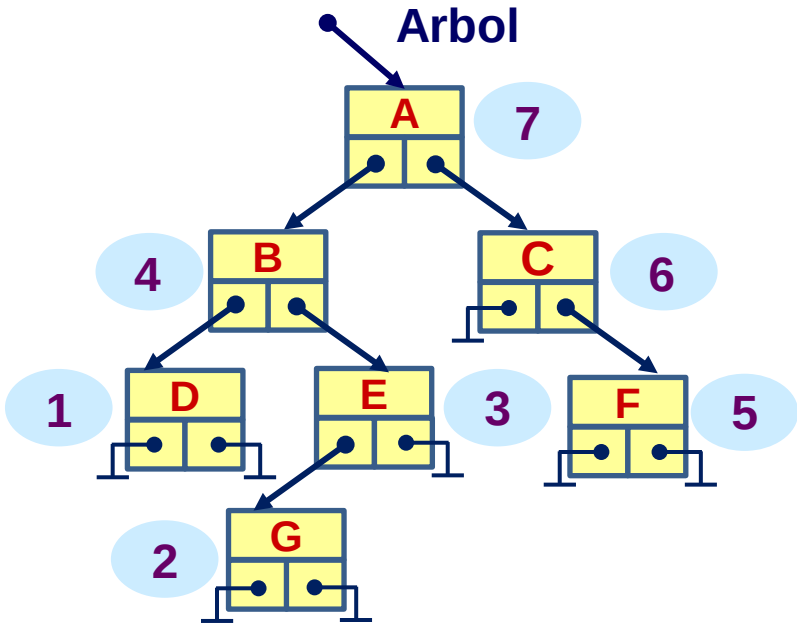
- **Recorrer el subárbol izquierdo (I)**
- **Recorrer el subárbol derecho (D)**
- **Procesar el nodo raíz (R)**



D, G, E, B, F, C, A

```
void Arbol::PostOrden(shared_ptr<Nodo> A) const {
    if (A != nullptr) {
        PostOrden (A->getHizq());      // I
        PostOrden (A->getHdch());      // D
        A->procesarNodo();             // R
    }
}

void Arbol::recorrerPostOrden() const {
    if (!empty()) {
        PostOrden (raiz);
    }
}
```

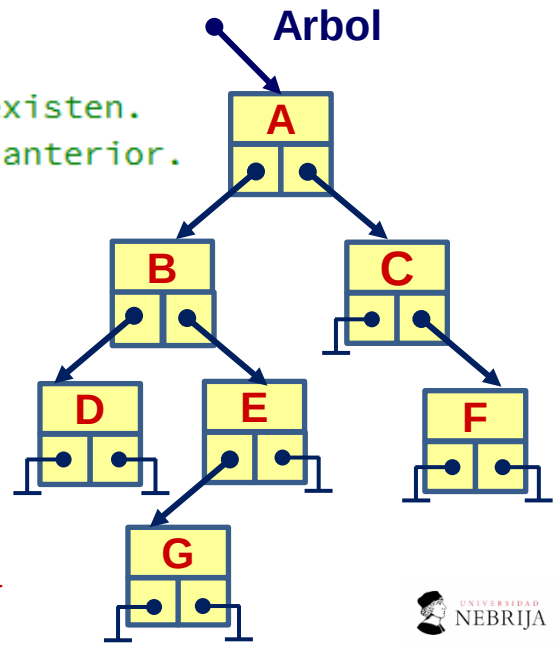
D, G, E, B, F, C, A



Se necesita una estructura Cola.

```
//---------------Clase NodoCola
class NodoCola {
    private:
        Arbol elem;
        shared_ptr<NodoCola> sig = nullptr;
    public:
        NodoCola():sig{nullptr} {};
        NodoCola(Arbol const &d, shared_ptr<NodoCola> ptr):elem(d),sig(ptr){}

        const Arbol &getElem() const;
        void setElem(const Arbol &newElem);

        const shared_ptr<NodoCola> &getSig() const;
        void setSig(const shared_ptr<NodoCola> &newSig);
};

//---------------Clase Cola
class Cola {
    private:
        shared_ptr<NodoCola> ppio;
        shared_ptr<NodoCola> final;

    public:
        Cola():ppio(nullptr), final(nullptr){};

        bool empty() const;
        void push(const Arbol &dato);
        void pop();
        Arbol first() const;
};
```

```
void Arbol::recorrerAnchura () const{
    Arbol Bi, Bd, B;
    Cola Niveles;
    CDato dato;

    if (!empty()) {
      Niveles.push(*this);    // Se añade el nodo raíz del árbol, 1er. Nivel
    }

    while(!Niveles.empty()) {
        B = Niveles.first();  // Se extrae un elemento de la cola
        Niveles.pop();

        dato = B.getDatoNodo();
        Bi.setRaiz(B.getHiNodo());
        Bd.setRaiz(B.getHdNodo());
        cout << "\n\tNodo: " << dato.getN();

        //  Se añaden al final de la cola el hi y hd, si es que existen.
        //  Siempre se procesan después de hacerlo los del nivel anterior.
        if (!Bi.empty()) {
            Niveles.push(Bi);
        }

        if (!Bd.empty()) {
            Niveles.push(Bd);
        }
    }
}
```



A, B, C, D, E, F, G

EJERCICIO DE ÁRBOLES

Escribir un programa que permita crear un árbol binario, como el de la figura, utilizando las operaciones estudiadas del TAD Árbol.

Después, una vez creado el árbol, realizar los diferentes recorridos en profundidad y en anchura mostrando el contenido de los nodos por pantalla.