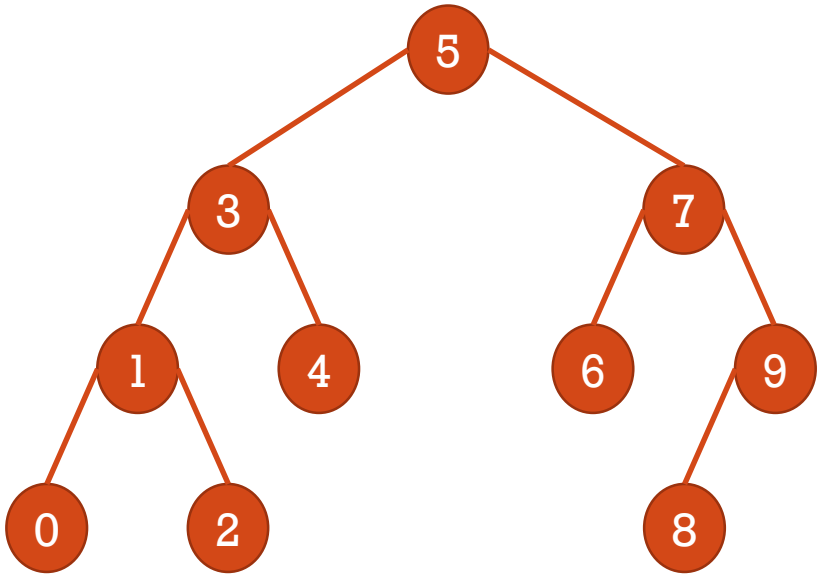
UNIDAD I. ESTRUCTURA DE DATOS BÁSICAS (ÁRBOLES BINARIOS DE BÚSQUEDA)

Francisco J. Hernández López

fcoj23@cimat.mx

ÁRBOLES BINARIOS DE BÚSQUEDA (ABB)

- Es un árbol que debe cumplir lo siguiente:
  - Para todo nodo T del árbol, todos los valores almacenados en el subárbol izquierdo de T son menores o iguales a la información guardada en el nodo T
  - Para todo nodo T del árbol, todos los valores almacenados en el subárbol derecho de T son mayores a la información guardada en el nodo T



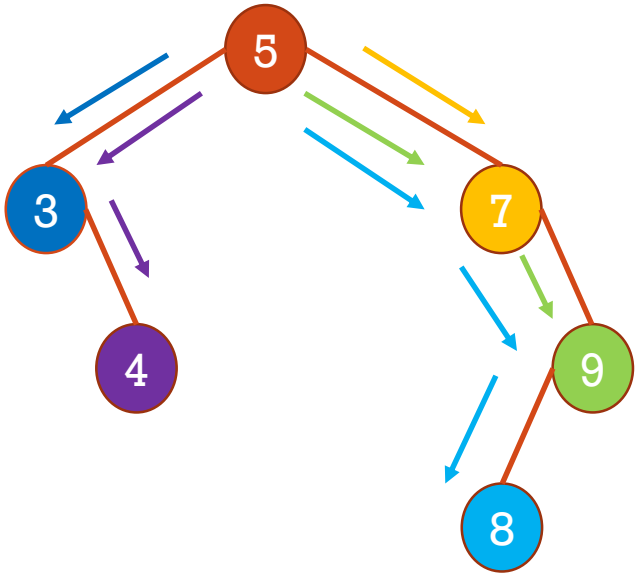
Su recorrido en In-orden nos genera los datos ordenados de forma ascendente:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

El orden de la búsqueda en un ABB es $\lceil \log_2(N+1) \rceil$ y N, con N el número de nodos del árbol

Estructura de datos, Cairó - Guardati, 3a. Edición, 2006.
Programación en C: metodología, algoritmos y estructura de datos, Luis Joyanes Aguilar, Ignacio Zahonero Martínez, 2a ed., 2005.

INSERTAR UN NUEVO ELEMENTO EN UN ABB



Insertar: 5
Cuando el árbol está vacío, entonces el primer elemento se convierte en la raíz

Insertar: 3

Insertar: 7

Insertar: 4

Insertar: 9

Insertar: 8

CÓDIGO PARA INSERTAR UN NUEVO ELEMENTO EN UN ABB

```
void insertar(tipoNodoArbol **raiz,int dato_nuevo){
    //Crear memoria
    tipoNodoArbol *nodo_nuevo;
    nodo_nuevo=(tipoNodoArbol *)malloc(sizeof(tipoNodoArbol));
    nodo_nuevo->dato=dato_nuevo;
    nodo_nuevo->der=NULL;
    nodo_nuevo->izq=NULL;

    if(*raiz==NULL){//El árbol está vacío
        (*raiz)=nodo_nuevo;
    }
    else{
        tipoNodoArbol *auxiliar;
        auxiliar=*raiz;
        int ban_insertar=0;//1-->Si se insertó el nodo
        while(ban_insertar==0){
            if(dato_nuevo>auxiliar->dato){//Si es mayor va hacia la derecha
                if(auxiliar->der!=NULL){
                    auxiliar=auxiliar->der;
                }
```

CÓDIGO PARA INSERTAR UN NUEVO ELEMENTO EN UN ABB (CONTINUACIÓN)

```
                else{//Insertamos el elemento
                    auxiliar->der=nodo_nuevo;
                    ban_insertar=1;//Ya insertamos el elemento
                }
            }
            else{//Si es menor o igual va hacia la izquierda
                if(auxiliar->izq!=NULL){
                    auxiliar=auxiliar->izq;
                }
                else{
                    auxiliar->izq=nodo_nuevo;
                    ban_insertar=1;//Ya insertamos el elemento
                }
            }
        }
    }
}
```

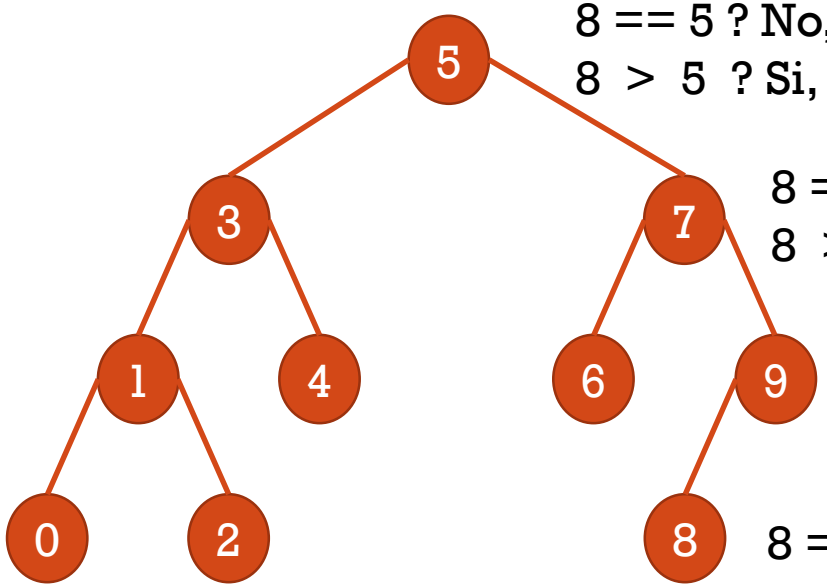
INSERTAR UN NUEVO ELEMENTO EN UN ABB DE FORMA RECURSIVA

```
void insertar_recursivo(tipoNodoArbol **raiz,int dato_nuevo){
    if(*raiz==NULL){//El árbol está vacío
        //Crear memoria
        tipoNodoArbol *nodo_nuevo;
        nodo_nuevo=(tipoNodoArbol *)malloc(sizeof(tipoNodoArbol));
        nodo_nuevo->dato=dato_nuevo;
        nodo_nuevo->der=NULL;
        nodo_nuevo->izq=NULL;
        (*raiz)=nodo_nuevo;
    }
    else{
        tipoNodoArbol *auxiliar;
        auxiliar=*raiz;

        if(dato_nuevo>auxiliar->dato){//Si es mayor va hacia la derecha
            insertar_recursivo(&auxiliar->der,dato_nuevo);
        }
        else{//Si es menor o igual va hacia la izquierda
            insertar_recursivo(&auxiliar->izq,dato_nuevo);
        }
    }
}
```

BÚSQUEDA

- En los ABB la búsqueda es más eficiente que en los árboles binarios generales
- Por ejemplo, si en el siguiente ABB queremos buscar el 8:



8 == 5 ? No,
8 > 5 ? Si, mover hacia la derecha

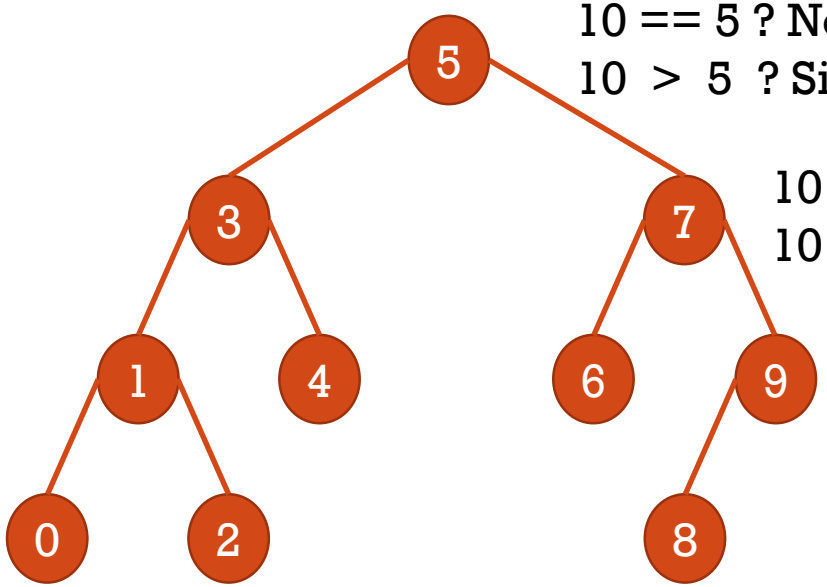
8 == 7 ? No,
8 > 7 ? Si, mover hacia la derecha

8 == 9 ? No,
8 > 9 ? No, mover hacia la izquierda

8 == 8 ? Si, "Encontrado…"

BÚSQUEDA

- En los ABB la búsqueda es más eficiente que en los árboles binarios generales
- Ahora queremos buscar el 10:



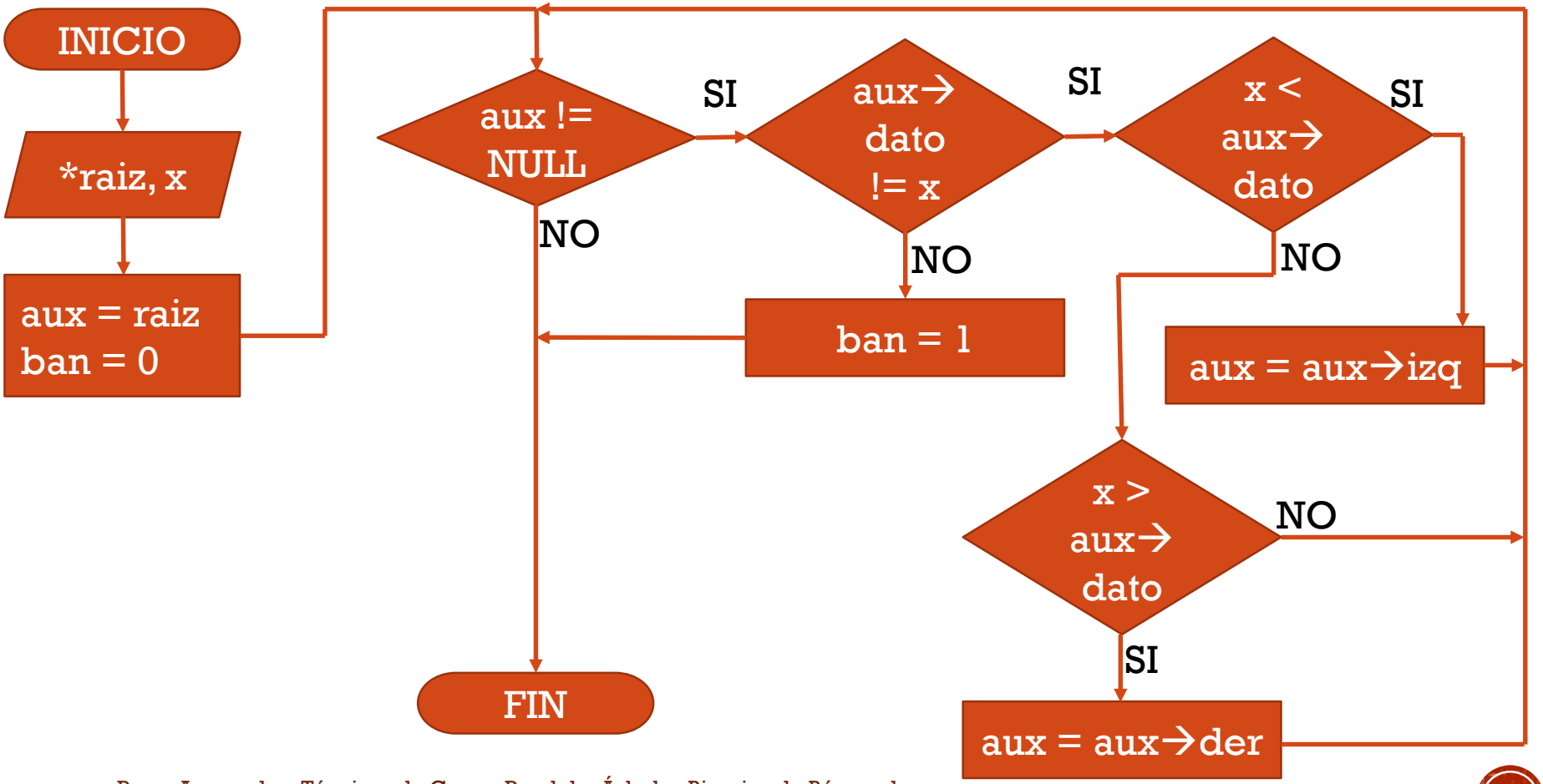
10 == 5 ? No,
10 > 5 ? Si, mover hacia la derecha

10 == 7 ? No,
10 > 7 ? Si, mover hacia la derecha

10 == 9 ? No,
10 > 9 ? Si, mover hacia la derecha

Como el nodo 9 apunta a NULL en su parte derecha, entonces la búsqueda finaliza: "No encontrado"

DIAGRAMA DE FLUJO PARA BÚSQUEDA DE UN DATO X EN UN ABB



VERSIÓN RECURSIVA DEL ALGORITMO DE BÚSQUEDA

```
int buscar_recursivo(tipoNodoArbol *raiz,int x){
    tipoNodoArbol *aux;
    int ban=0;
    aux=raiz;

    if(aux!=NULL){
        if(aux->dato != x){
            if(x < aux->dato){
                return buscar_recursivo(aux->izq,x);
            }
            else{
                return buscar_recursivo(aux->der,x);
            }
        }
        else{
            ban=1;
        }
    }
    return(ban);
}
```

Eliminar un elemento del ABB

- Para eliminar un nodo del ABB, hay que tener cuidado con los siguientes casos:
  - Si el elemento a eliminar es un nodo hoja, simplemente se elimina redefiniendo el puntero de su padre
  - Si el elemento a eliminar tiene un nodo hijo, entonces tiene que sustituirse por ese hijo
  - Si el elemento a eliminar tiene los dos hijos, entonces se tiene que sustituir por el nodo que se encuentra más a la izquierda en el subárbol derecho o por el nodo que se encuentra más a la derecha del subárbol izquierdo

CÓDIGO EN C PARA ELIMINAR UN ELEMENTO DE UN ABB

```
void eliminar(tipoNodoArbol **raiz,int dato_a_eliminar){
    tipoNodoArbol *aux,*padre;
    aux=*raiz;//Para buscar el nodo a eliminar
    padre=aux;//Va a ser el padre del nodo a eliminar
    if(aux==NULL){
        printf("\nArbol vacio...");
    }
    else{
        //Buscar el elemento a eliminar
        int ban=0;
        while(aux!=NULL){
            if(aux->dato != dato_a_eliminar){
                if(dato_a_eliminar < aux->dato){
                    padre=aux;
                    aux=aux->izq;
                }
                else{
                    padre=aux;
                    aux=aux->der;
                }
            }
            else{
                ban=1;
                break;
            }
        }
```

```
if(ban==1){//Si se encuentra el elemento
    tipoNodoArbol *temp;
    temp=aux;
    //Caso 1. No tiene hijos
    if(aux->der==NULL && aux->izq==NULL){
        if(padre==*raiz)
            *raiz=NULL;//Árbol vacío
        else{
            if(padre->izq==aux)
                padre->izq=NULL;
            else
                padre->der=NULL;
        }
    }
    else{//Caso 2. Tiene solo un hijo
        if(aux->der==NULL){//No tiene el hijo derecho
            if(padre->izq==aux)
                padre->izq=aux->izq;
            else
                padre->der=aux->izq;
        }
        else{
            if(aux->izq==NULL){//No tiene el hijo izquierdo
                if(padre->izq==aux)
                    padre->izq=aux->der;
                else
                    padre->der=aux->der;
            }
```

Código en C para eliminar un elemento de un ABB (continuación)

```
                else{//Caso 3. Tiene dos hijos
                    //Buscar el nodo más a la izq del subárbol derecho
                    tipoNodoArbol *nodo_izq_SD;
                    nodo_izq_SD=aux->der;
                    while(nodo_izq_SD->izq!=NULL){
                        padre=nodo_izq_SD;
                        nodo_izq_SD=nodo_izq_SD->izq;
                    }
                    //Reemplazar nodo
                    aux->dato=nodo_izq_SD->dato;
                    if(padre->izq==nodo_izq_SD)
                        padre->izq=nodo_izq_SD->der;
                    else
                        padre->der=nodo_izq_SD->der;
                    aux=nodo_izq_SD;
                }
            }
        }
        //Liberar memoria
        free(aux);
    }
    else{
        printf("\nNo se encontró el elemento a eliminar...");
    }
  }
}
```

VERSIÓN RECURSIVA PARA ELIMINAR UN ELEMENTO DEL ABB

```
void eliminar_recursivo(tipoNodoArbol **raiz,int dato_a_eliminar){
    tipoNodoArbol *aux,*padre;
    aux=*raiz;
    if(aux!=NULL){
        //Buscar el elemento a eliminar
        if(aux->dato != dato_a_eliminar){
            if(dato_a_eliminar < aux->dato){
                eliminar_recursivo(&aux->izq,dato_a_eliminar);
            }
            else{
                eliminar_recursivo(&aux->der,dato_a_eliminar);
            }
        }
        else{
            if(aux->izq == NULL){
                *raiz=(*raiz)->der;
            }
            else{
                if(aux->der==NULL){
                    *raiz=(*raiz)->izq;
                }
                else{
                    reemplazar(&(*raiz)->izq,&aux);
                }
            }
            free(aux);
        }
    }
}
```

Nota: En esta versión, cuando el nodo a eliminar contiene dos hijos, entonces lo reemplazamos por el nodo que está más a la derecha del subárbol izquierdo

```
void reemplazar(tipoNodoArbol **raiz,tipoNodoArbol **aux){
    if((*raiz)->der==NULL){
        (*aux)->dato=(*raiz)->dato;
        (*aux)=*raiz;
        *raiz=(*raiz)->izq;
    }
    else{
        reemplazar(&(*raiz)->der,aux);
    }
}
```

ABB EQUILIBRADOS

- Las alturas de los dos subárboles de cada nodo tiene como máximo una diferencia de 1 en valor absoluto, es decir el Factor de Equilibrio: $|FE| \leq 1$ en cada nodo
- Entonces cuando se realizan las operaciones: insertar y borrar
  - Pueden ocasionar que el ABB quede desequilibrado
  - Para volver a equilibrar el ABB hay que realizar ciertos movimientos (rotaciones)
- De acuerdo al criterio tomado para realizar el equilibrado existen varios tipos de árboles ABB:
  - AVL
  - Rojo-Negro
  - AA
  - Etc…