Algoritmos y Estructura de datos

Clase 12
Arboles parte 2

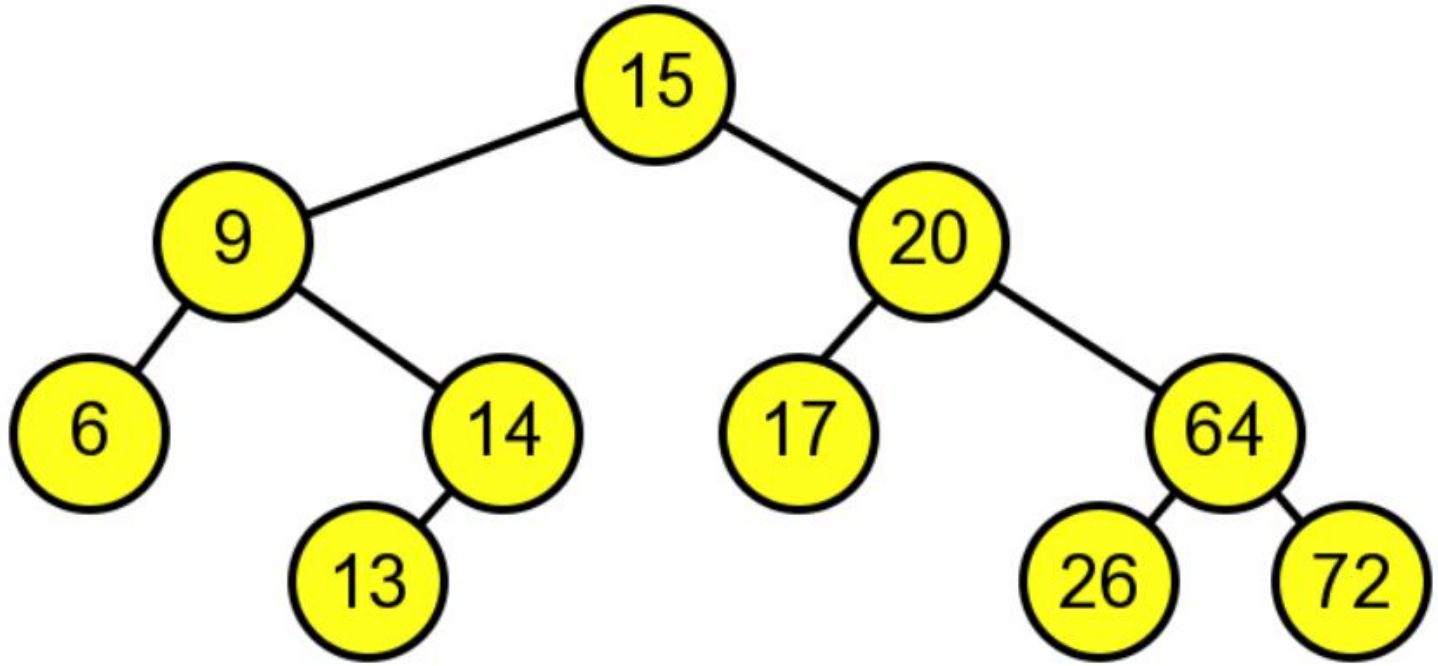
Profesor: Carlos Diaz

Contenido

- Definición de Arbol binario de búsqueda (Binary search tree - BST)
- Creación de un BST
- La clase Nodo
- La clase ArbolBB
- La función crear() no recursivo
- La función completarArbol()
- Tipos de recorrido en un árbol binario
- Código recursivo de los tipos de recorrido en profundidad
- La función crear() recursiva
- Ejercicios

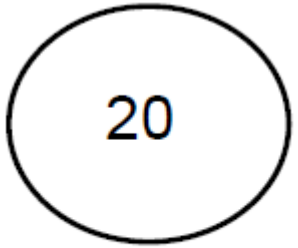
Definición de un árbol binario de búsqueda

- Un árbol binario de búsqueda es aquel que dado un nodo, todos los datos del subárbol izquierdo son menores que el dato de ese nodo, mientras que todos los datos del subárbol derecho son mayores que ese dato.

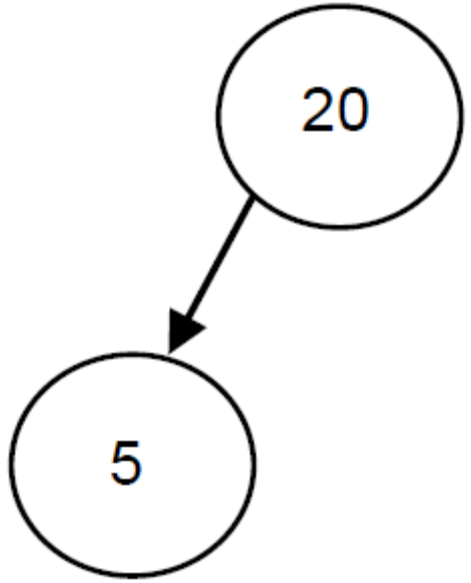


Creación un de árbol binario de búsqueda

- Sean los elementos 20, 5, 9, 40, 70, 60.
- El primer elemento de la lista se asigna al nodo raíz, según los valores, los elementos se colocan a la izquierda o al lado derecho de la raíz.
- Como se ve en la figura a, primero se crea el nodo raíz y tiene el valor de 20 y su nodo izquierdo y derecho apuntarán al NULL.
- Los otros elementos se comparan con este nodo. Elemento mayor al nodo padre, se coloca a la derecha, de lo contrario se coloca a la izquierda.
- En consecuencia, figura b, la llegada del siguiente elemento, es decir, 5, se comparará con el nodo raíz (20), ya que es menor que el nodo raíz, se coloca a la izquierda, pues no hay ningún nodo a la izquierda de 20.



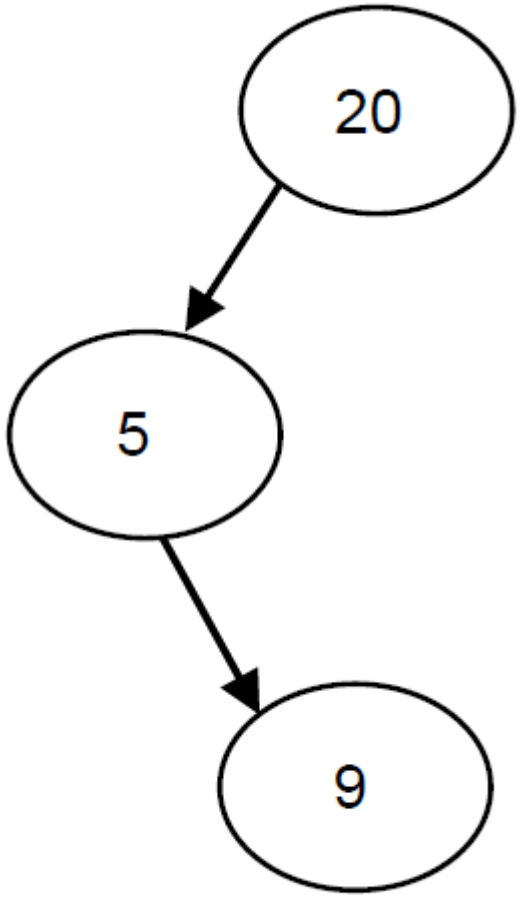
a.



b.

Creación un de árbol binario de búsqueda

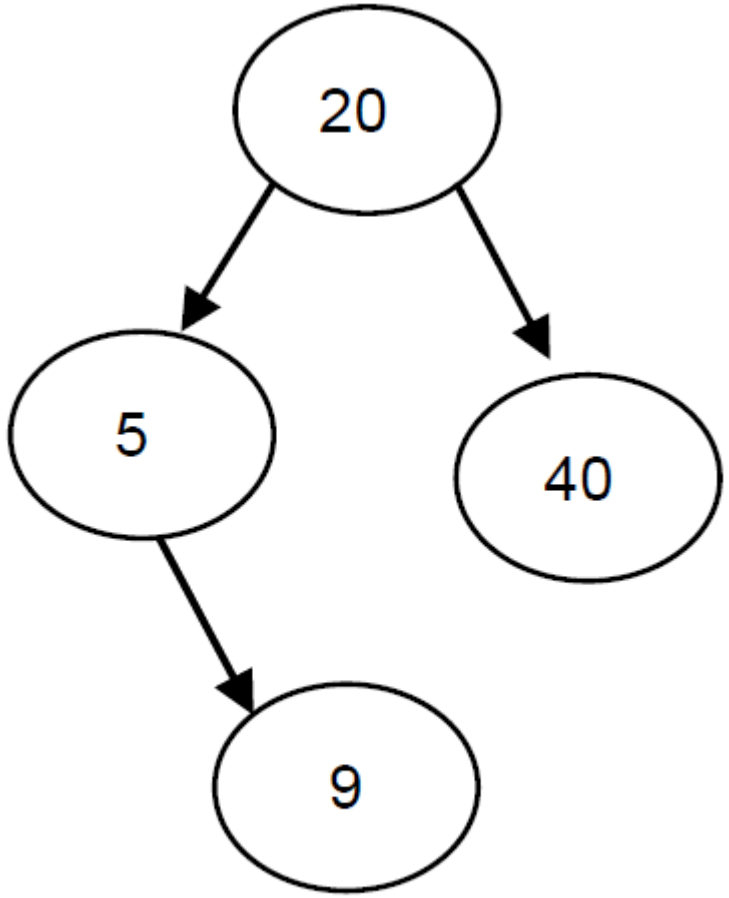
- El siguiente número que llega es 9, nuevamente la comparación comienza desde el nodo raíz (20), ya que 9 es menor que 20, se dirigirá hacia la izquierda, después de dirigirse un nivel hacia abajo a la izquierda, alcanza el nodo con valor 5, ya que, 9 es mayor que 5, se dirigirá hacia la derecha de 5, ya que no hay ningún nodo a la derecha de 5, entonces este nodo se colocara a la derecha de 5, tal como se ve en la figura c.



c.

Creación un de árbol binario de búsqueda

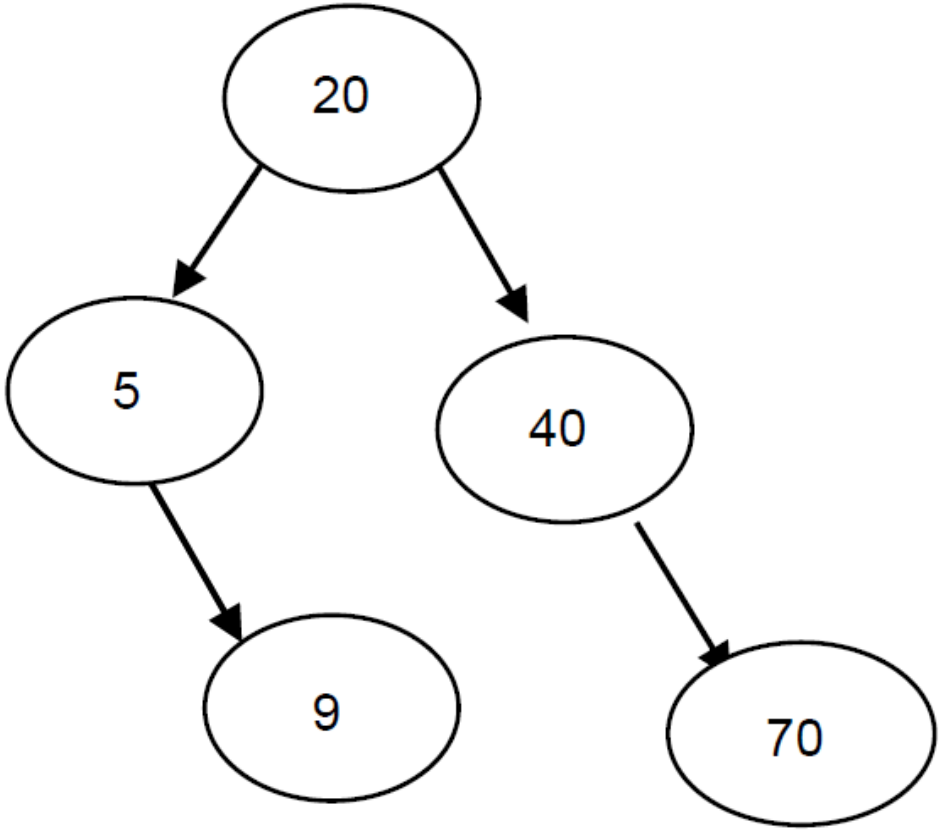
- El siguiente número de la serie es 40, nuevamente la comparación comenzará desde la raíz, es decir, 20, como 40 es mayor que 20 se dirigirá hacia la derecha de 20, ya que el árbol no tiene nodo a la derecha de 20, se coloca allí el nodo de 40, figura d.



d.

Creación un de árbol binario de búsqueda

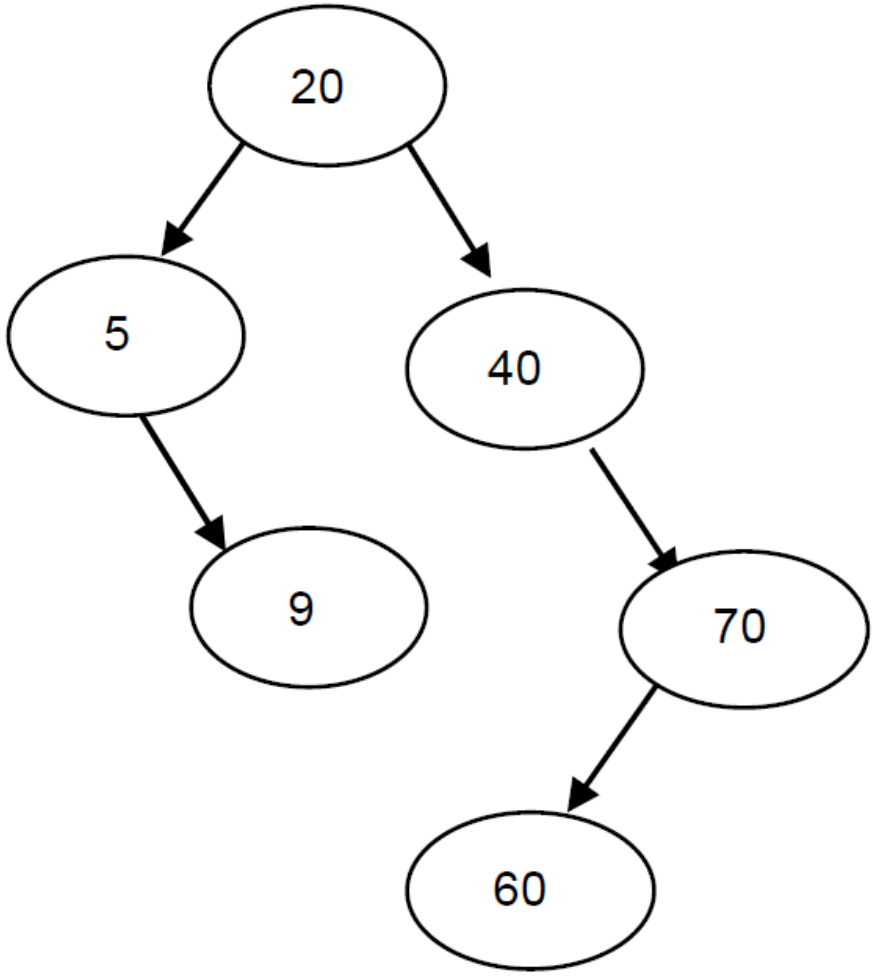
- El siguiente nodo a ser insertado en el árbol es el de valor 70.
- Correspondiente al nodo anterior, la comparación comenzará desde el nodo raíz (20), ya que, 70 es más que 20, se dirigirá hacia el lado derecho, bajando un nivel, se compara nuevamente con el nodo que tiene valor 40, ya que aún es mayor, se dirigirá hacia la derecha de este nodo también, ya que su derecha no tiene nodo, se adjuntará allí el nodo de valor 70, figura e.



e.

Creación un de árbol binario de búsqueda

- El ultimo nodo es el de 60.
- La comparación se reiniciará desde el nodo raíz (20), ya que es más de 20, se dirigirá hacia la derecha de este nodo, se encuentra 40, ya que 60 es más que el 40, se dirige hacia la derecha.
- Ahora, se encontrara a 70, ya que 60 es menos que 70, se dirigirá hacia la izquierda. Dado que la izquierda no hay nodo, allí de coloca el nodo de valor 60, figura f.



f.

La clase Nodo

- Primero utilizaremos un programa no recursivo.
- Definimos la clase Nodo con tres elementos, es decir, el puntero izquierdo, el puntero derecho y el dato.
- Inicialmente apuntan a NULL.

```
#include <iostream>
using namespace std;
class Nodo
{
        public:
        int dato;
        Nodo *izq, *der;
        Nodo(){
                dato=NULL;
                izq=der=NULL;
        }
};
```

La clase arbolBB

- La clase arbolBB crea el nodo raíz y lo inicializa a NULL.
- Define el método crear() que inserta un nodo en el árbol.

```
class arbolBB
{
        public:
        Nodo *raiz;
        arbolBB()
        {
                raiz=NULL;
        }
        void crear(int);
};
```

La función crear() no recursiva

```
void arbolBB::crear(int nuevoDato)
{
    //Crea la raíz del arbol
    if(raiz==NULL)
    {
        raiz=new Nodo;
        raiz->dato=nuevoDato;
        raiz->izq=raiz->der=NULL;
        return;
    }
    //Crea un nodo y le asigna el nuevo dato, en el caso que ya exista raíz.
    Nodo *nuevoNodo,*r;
    nuevoNodo =new Nodo;
    nuevoNodo ->dato=nuevoDato;
    nuevoNodo ->izq= nuevoNodo ->der=NULL;
```

La función crear() no recursiva

```
//Inicializa r con la raiz
r=raiz;
while(r!=NULL) //Continua mientras la raíz no sea una hoja
{
        // Si el nuevoDato es menor que el dato raíz entonces se mueve a la izquierda
        if(nuevoDato < r->dato)
        {
                // Si izq is NULL coloca el nuevoNodo a la izquierda.
                if(r->izq==NULL)
                {
                        r->izq=nuevoNodo;
                        break;
                }
                        r=r->izq; // Si no mueve la raíz al nodo de la izquierda
        }
```

La función crear() no recursiva

```
            // Si el nuevoDato es mayor que el dato raíz entonces se mueve a la derecha
            else if(nuevoDato > r->dato)
            {
                        // Si der is NULL coloca el nuevoNodo a la derecha.
                        if(r->der==NULL)
                        {
                                    r->der=nuevoNodo;
                                    break;
                        }
                                    r=r->der; // Si no mueve la raíz al nodo de la derecha
            }
      }
}
```

La función completarArbol()

- La función main() llama a la función completarArbol().

```
int main()
{
        arbolBB a;
        completarArbol(a);
        return 0;
}
```

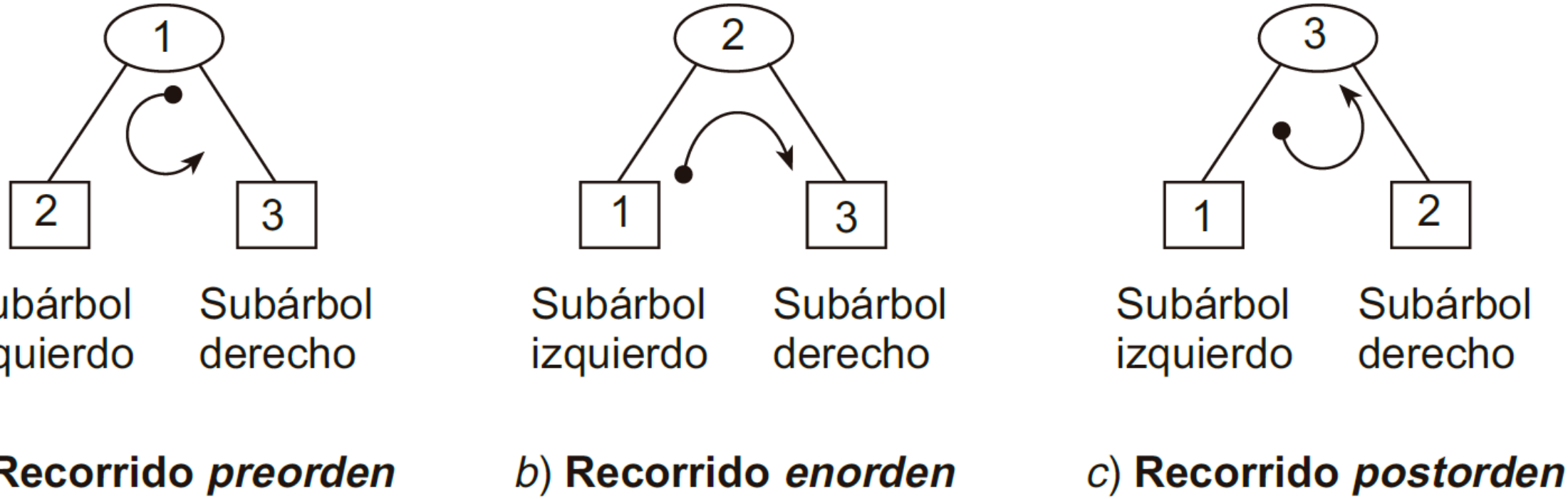
```
void completarArbol(arbolBB &a)
{
        int n,dato;
        cout<<"Cuantos nodos desea insertar: ";
        cin>>n;
        for (int i=0;i<n;i++)
        {
                cout<<"Dato "<<i<<": ";
                cin>>dato;
                a.crear(dato);
        }
}
```

Recorrido en un Arbol binario

- Para visualizar o consultar los datos almacenados en un árbol se necesita recorrer el árbol o *visitar* los nodos del mismo.
- El recorrido de un árbol binario requiere que cada nodo del árbol sea procesado (visitado) una vez y sólo una en una secuencia predeterminada. Existen dos enfoques generales para la secuencia de recorrido, profundidad y anchura.
- En el recorrido en profundidad, el proceso exige un camino desde el raíz a través de un hijo, al descendiente más lejano del primer hijo antes de proseguir a un segundo hijo. En otras palabras, en el recorrido en profundidad, todos los descendientes de un hijo se procesan antes del siguiente hijo.
- En el recorrido en anchura, el proceso se realiza horizontalmente desde el raíz a todos sus hijos, a continuación a los hijos de sus hijos y así sucesivamente hasta que todos los nodos han sido procesados. En el recorrido en anchura, cada nivel se procesa totalmente antes de que comience el siguiente nivel.

Tipos de recorrido en profundidad

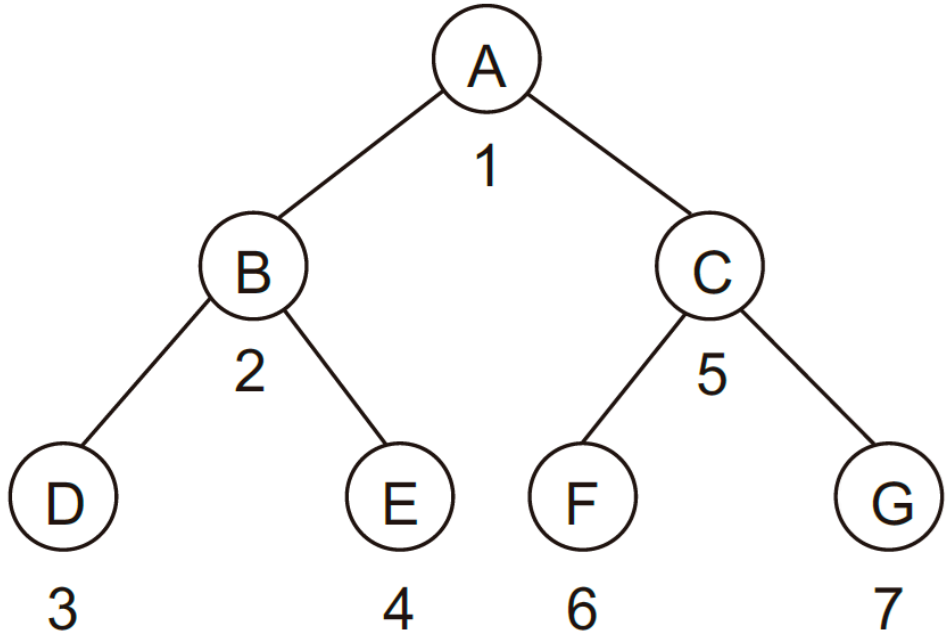
- Tenemos tres tipos de recorrido:
  1. Recorrido preorden
  2. Recorrido enorden
  3. Recorrido postorden



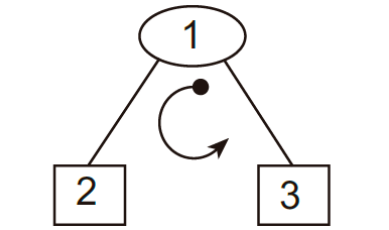
a) **Recorrido *preorden*** *b*) **Recorrido *enorden*** *c*) **Recorrido *postorden***

Recorrido preorden

- El recorrido preorden conlleva los siguientes pasos, en los que el nodo raíz va antes que los subárboles:
  1. Visitar el nodo raíz (**N**).
  2. Recorrer el subárbol izquierdo (**I**) en preorden.
  3. Recorrer el subárbol derecho (**D**) en preorden.
- El algoritmo de recorrido tiene naturaleza recursiva. Primero, se procesa la raíz, a continuación el subárbol izquierdo y a continuación el subárbol derecho.
- Para procesar el subárbol izquierdo se siguen los mismos pasos: raíz, subárbol izquierdo y subárbol derecho (proceso recursivo).
- Luego se hace lo mismo con el subárbol derecho.

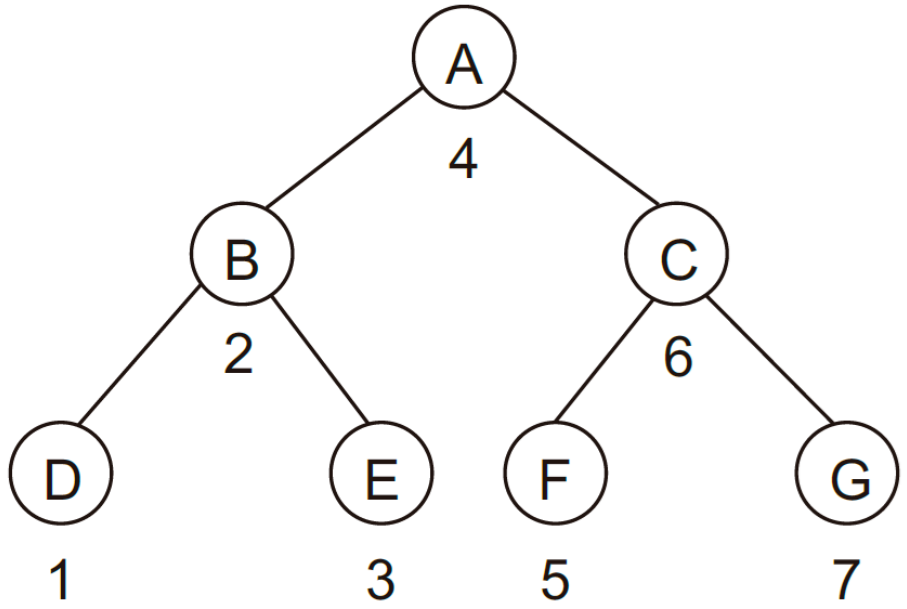


Recorrido: A, B, D, E, C, F, G

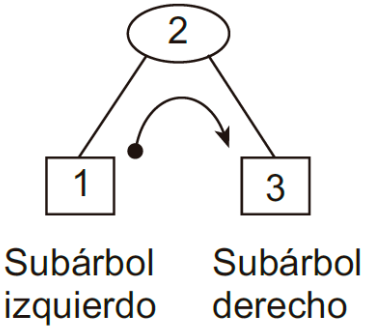


Recorrido enorden

- El recorrido enorden procesa primero el subárbol izquierdo, después el raíz y a continuación el subárbol derecho.
- Si el árbol no está vacío, el método implica los siguientes pasos:
  1. Recorrer el subárbol izquierdo (**I**) en enorden.
  2. Visitar el nodo raíz (**N**).
  3. Recorrer el subárbol derecho (**D**) en enorden.

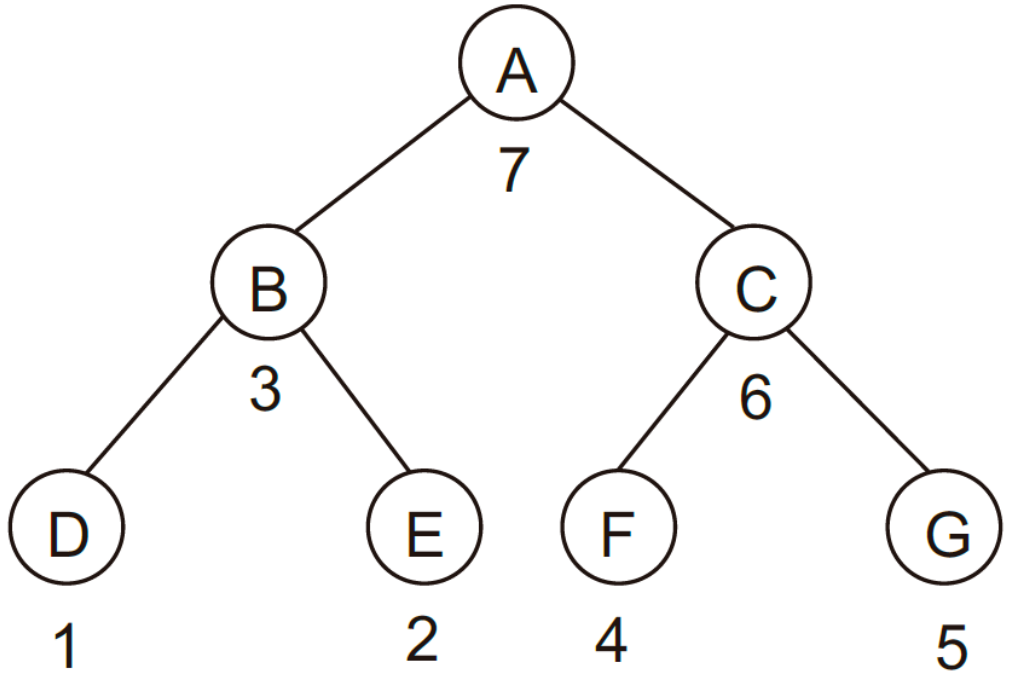



```
Recorrido: D, B, E, A, F, C, G
```

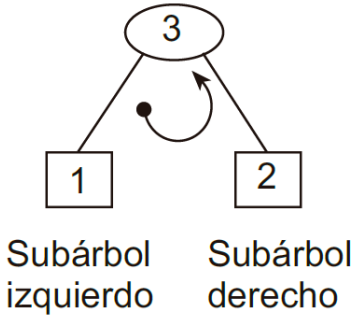


Recorrido postorden

- El recorrido postorden procesa el nodo raíz después de que los subárboles izquierdo y derecho se han procesado. Se comienza situándose en la hoja más a la izquierda y se procesa. A continuación se procesa su subárbol derecho. Por último, se procesa el nodo raíz.
- Si el árbol no está vacío, el método implica los siguientes pasos:
  1. Recorrer el subárbol izquierdo (**I**) en postorden.
  2. Recorrer el subárbol derecho (**D**) en postorden.
  3. Visitar el nodo raíz (**N**).



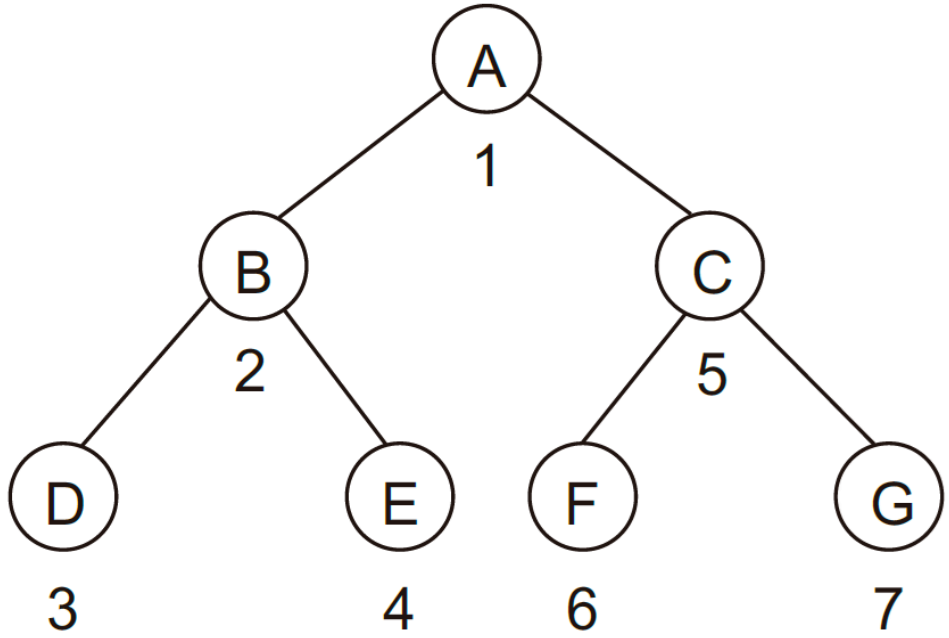

```
Recorrido: D, E, B, F, G, C, A
```



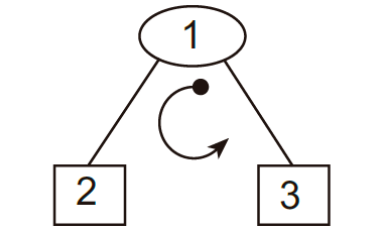
Código recorrido preorden

1. Visitar el nodo raíz (**N**).
2. Recorrer el subárbol izquierdo (**I**) en preorden.
3. Recorrer el subárbol derecho (**D**) en preorden.

```
//Funcion recursiva para preorden
void arbolBB::preOrden (Nodo *raiz)
{
        if (raiz!=NULL)
        {
                cout<<raiz->dato<<" ";
                preOrden(raiz->izq);
                preOrden(raiz->der);
        }
}
```



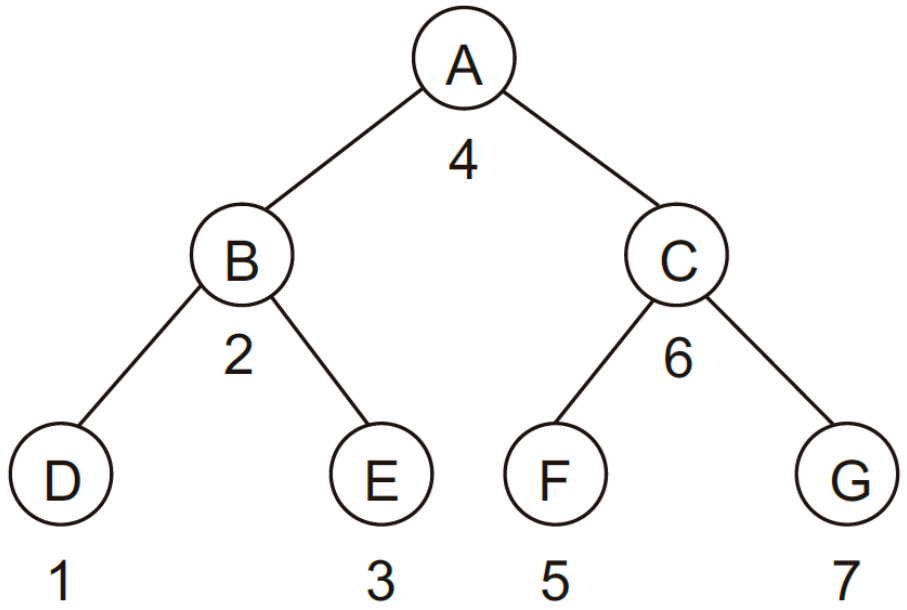
Recorrido: A, B, D, E, C, F, G



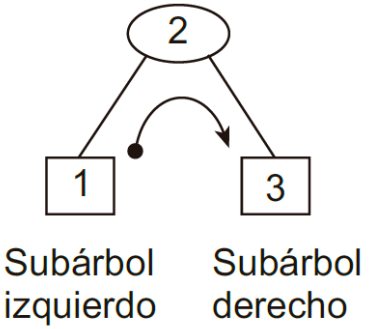
Recorrido enorden

1. Recorrer el subárbol izquierdo (**I**) en enorden.
2. Visitar el nodo raíz (**N**).
3. Recorrer el subárbol derecho (**D**) en enorden.

```
//Funcion recursiva para enorden
void arbolBB::enOrden(Nodo *raiz)
{
        if (raiz!=NULL)
        {
                enOrden (raiz->izq);
                cout<<raiz->dato<< "  ";
                enOrden (raiz->der);
        }
}
```



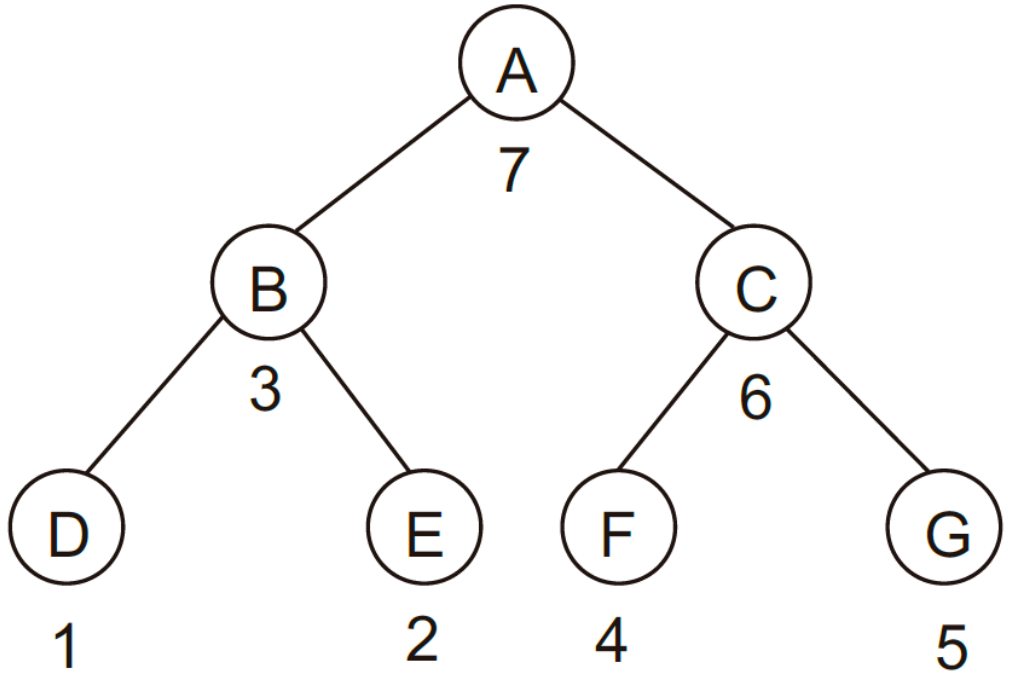

```
Recorrido: D, B, E, A, F, C, G
```



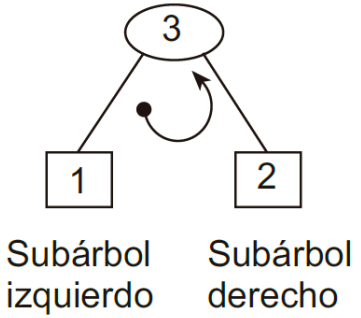
Recorrido postorden

1. Recorrer el subárbol izquierdo (**I**) en postorden.
2. Recorrer el subárbol derecho (**D**) en postorden.
3. Visitar el nodo raíz (**N**).

```
//Funcion recursiva para postorden
void arbolBB::postOrden(Nodo *raiz)
{
        if(raiz!=NULL)
        {
                postOrden(raiz->izq);
                postOrden(raiz->der);
                cout<<raiz->dato<<" ";
        }
}
```

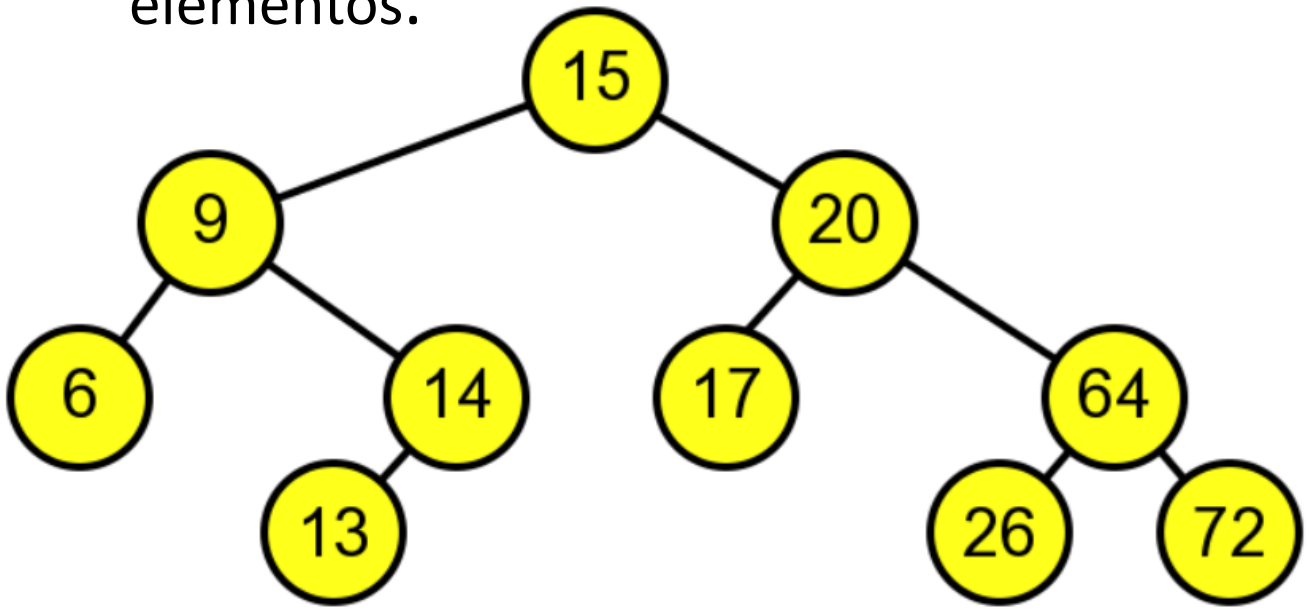


Recorrido: D, E, B, F, G, C, A



Ejemplo 1

- Verificar el programa BST no recursivo de 10 elementos.



Preorden = [15, 9, 6, 14, 13, 20, 17, 64, 26, 72]

Inorden = [6, 9, 13, 14, 15, 17, 20, 26, 64, 72]

Postorden =[6, 13, 14, 9, 17, 26, 72, 64, 20, 15]

```
int main()
{
        arbolBB a;
        completarArbol(a);
        cout<<"\npreorden"<<endl;
        a.preOrden(a.raiz);
        cout<<"\nenorden"<<endl;
        a.enOrden(a.raiz);
        cout<<"\nposorden"<<endl;
        a.postOrden(a.raiz);
        return 0;
}
```

La función crear() recursiva

```
// Funcion para la llamada recursiva
void arbolBB::crear(int nuevoDato)
{
        if (raiz==NULL) //Si no tiene raíz, crea el nodo raiz
        {
                raiz=new Nodo;
                raiz->dato=nuevoDato;
                return;
        }
                Nodo *r=raiz;
                crear(r, nuevoDato); //Llamada a la función recursiva
}
```

La función crear() recursiva

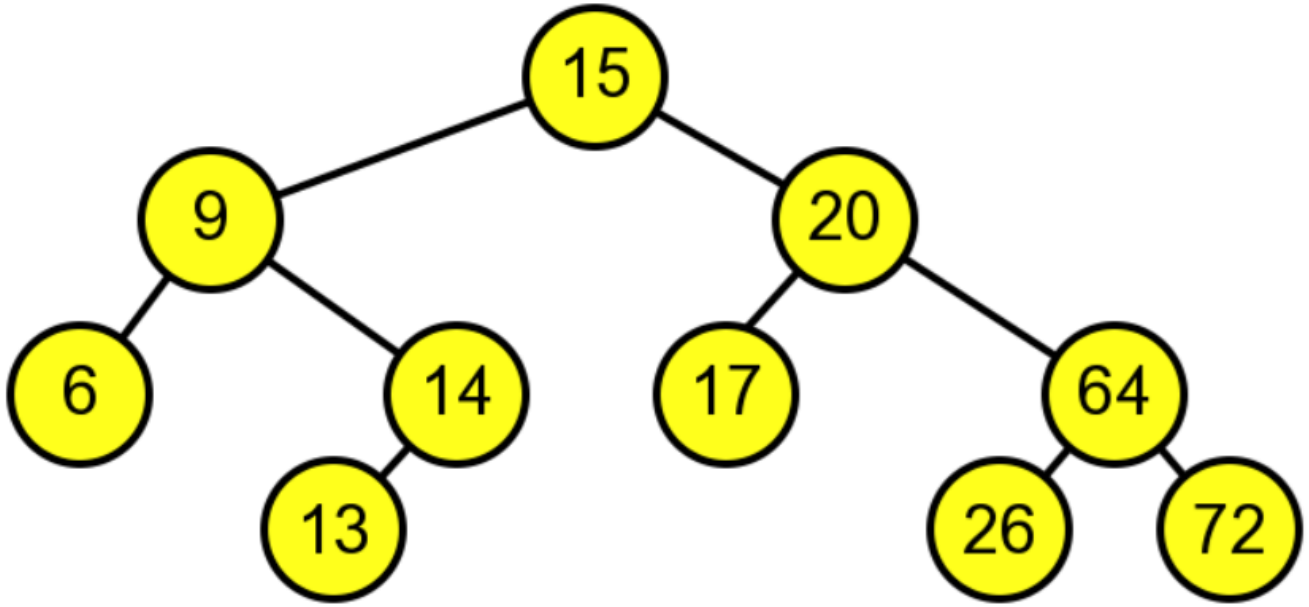
```
// Funcion recursiva
void arbolBB::crear(Nodo *r, int nuevoDato)
{
        if (nuevoDato < r->dato) // Si el nuevoDato es menor que el dato raíz entonces se mueve a la izquierda
        {
                if(r->izq==NULL) // Si izq is NULL coloca el nuevoNodo a la izquierda.
                {
                        Nodo *nuevoNodo;
                        nuevoNodo=new Nodo;
                        nuevoNodo->dato=nuevoDato;
                        r->izq=nuevoNodo;
                        return;
                }
                crear(r->izq, nuevoDato); // Si no mueve la raíz al nodo de la izquierda
        }
```

La función crear() recursiva

```
        else
        {
                if (nuevoDato> r->dato) // Si el nuevoDato es mayor que el dato raíz entonces se mueve a la derecha
                {
                        if(r->der==NULL) // Si der is NULL coloca el nuevoNodo a la derecha.
                        {
                                Nodo *nuevoNodo=new Nodo;
                                nuevoNodo->dato=nuevoDato;
                                r->der=nuevoNodo;
                                return;
                        }
                        crear(r->der, nuevoDato); // Si no mueve la raíz al nodo de la derecha
                }
        }
}
```

Ejemplo 2

- Verificar el programa BST recursivo.



Preorden = [15, 9, 6, 14, 13, 20, 17, 64, 26, 72]

Inorden = [6, 9, 13, 14, 15, 17, 20, 26, 64, 72]

Postorden =[6, 13, 14, 9, 17, 26, 72, 64, 20, 15]

```
int main()
{
        arbolBB a;
        completarArbol(a);
        cout<<"\npreorden"<<endl;
        a.preOrden(a.raiz);
        cout<<"\nenorden"<<endl;
        a.enOrden(a.raiz);
        cout<<"\nposorden"<<endl;
        a.postOrden(a.raiz);
        return 0;
}
```

FIN