Árboles

Ing José Luis MARTÍNEZ

6 de junio de 2019

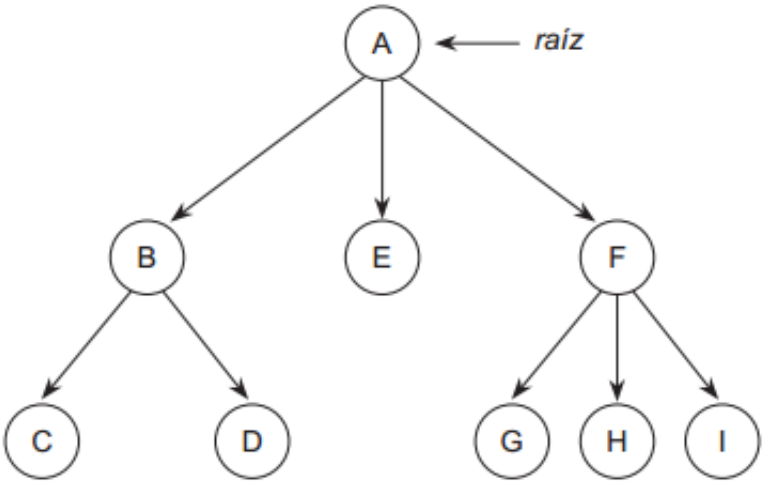
1. Introducción

- Los árboles son estructuras de datos no lineales al contrario que los arrays y las listas enlazadas que constituyen estructuras lineales. Dentro de las listas enlazadas incluimos las pilas y las colas.
- En las listas enlazadas cada nodo está comunicado en forma secuencial con el siguiente y el anterior
- Un árbol es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos.

1.1. Árboles informáticos generales

Un árbol es un tipo de dato estructurado que tiene una estructura jerárquica entre sus elementos. Un árbol es un conjunto de uno o nodos tales que:

- Hay un nodo diseñado especialmente llamado raíz.
- Los nodos restantes se dividen en $n \geq 0$ conjuntos disjuntos tales que $T_1 \dots T_n$, son un árbol. A $T_1, T_2, \dots T_n$ se les denomina subárboles del raíz.
- Si un árbol no está vacío, entonces el primer nodo se llama raíz.



1.2. Terminología en árboles

- El primer nodo de un árbol, normalmente dibujado en la posición superior, se denomina raíz del árbol.
- Las flechas que conectan un nodo a otro se llaman arcos o ramas.
- Los nodos terminales, (nodos de los cuales no se deduce ningún nodo), se denominan hojas.

- Los nodos que no son hojas se denominan nodos internos o nodos no terminales.
- Si en un árbol una rama va de un nodo n_1 a un nodo n_2, se dice que n_1 es el padre de n_2 y que n_2 es un hijo de n_1.
- n_1 se llama ascendiente de n_2 si n_1 es el padre de n_2 o si n_1 es el padre de un ascendiente de n_2.
- n_2 se llama descendiente de n_1 si n_1 es un ascendiente de n_2.
- Un camino de n_1 a n_2 es una secuencia de arcos contiguos que van de n_1 a n_2.
- La longitud de un camino es el número de arcos que contiene (en otras palabras, el número de nodos –1).
- El nivel de un nodo es la longitud del camino que lo conecta al raíz.
- La profundidad o altura de un árbol es la longitud del camino más largo que conecta el raíz a una hoja.
- Un subarbol de un árbol es un subconjunto de nodos del árbol, conectados por ramas del propio árbol, esto es a su vez un árbol.
- Sea SA un subárbol de un árbol A: si para cada nodo n de SA, SA contiene también todos los descendientes de n en A. SA se llama un subárbol completo de A.
- Un árbol está equilibrado cuando, dado un número máximo K de hijos de cada nodo y la altura del árbol h, cada nodo de nivel $k < h$–1 tiene exactamente K hijos.

2. Árboles binarios

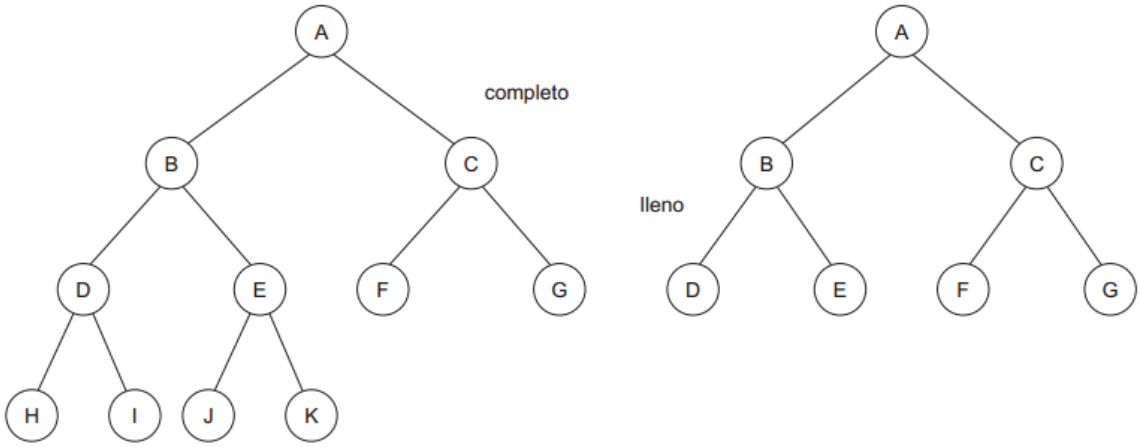
1 Un *árbol binario* es aquél en que cada nodo tiene como máximo grado dos.

2 Un árbol binario es un árbol en el que ningún nodo puede tener más de dos subárboles.

3 En un árbol binario, cada nodo puede tener, cero, uno o dos hijos (subárboles).

4 Se conoce el nodo de la izquierda como *hijo izquierdo* y el nodo de la derecha como *hijo derecho*.

- Un árbol binario está perfectamente equilibrado, si los subárboles de todos los nodos tienen la misma altura.
- Un árbol binario se dice que es completo si todos los nodos interiores, es decir aquéllos con descendientes, tienen dos hijos.
- Un árbol binario se dice lleno si todas sus hojas están al mismo nivel y todo sus nodos interiores tienen cada uno dos hijos. Si un árbol binario es lleno entonces es completo.



2.1. Operaciones que se realizan con árboles binarios

- Agregar nodos.
- Quitar nodos
- Recorrido del árbol.
- Búsqueda.

2.2. Recorridos de un árbol

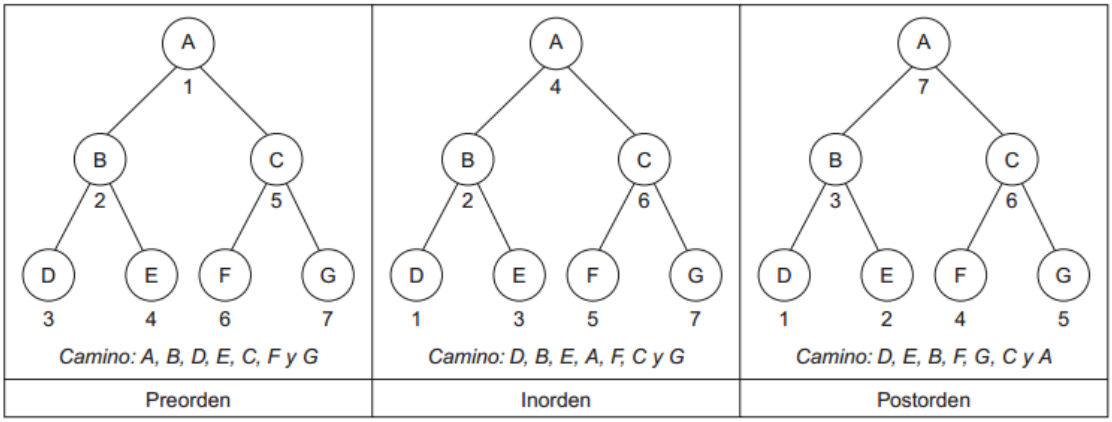
Para visualizar o consultar los datos almacenados en un árbol se necesita recorrer el árbol o visitar los nodos del mismo. Se denomina recorrido al proceso que permite acceder una sola vez a cada uno de los nodos del árbol. Existen diversas formas de efectuar el recorrido de un árbol binario:

1 **Recorrido en anchura:** Consiste en recorrer los distintos niveles (del inferior al superior), y dentro de cada nivel, los diferentes nodos de izquierda a derecha (o bien de derecha a izquierda).

2 **Recorrido en profundidad:**

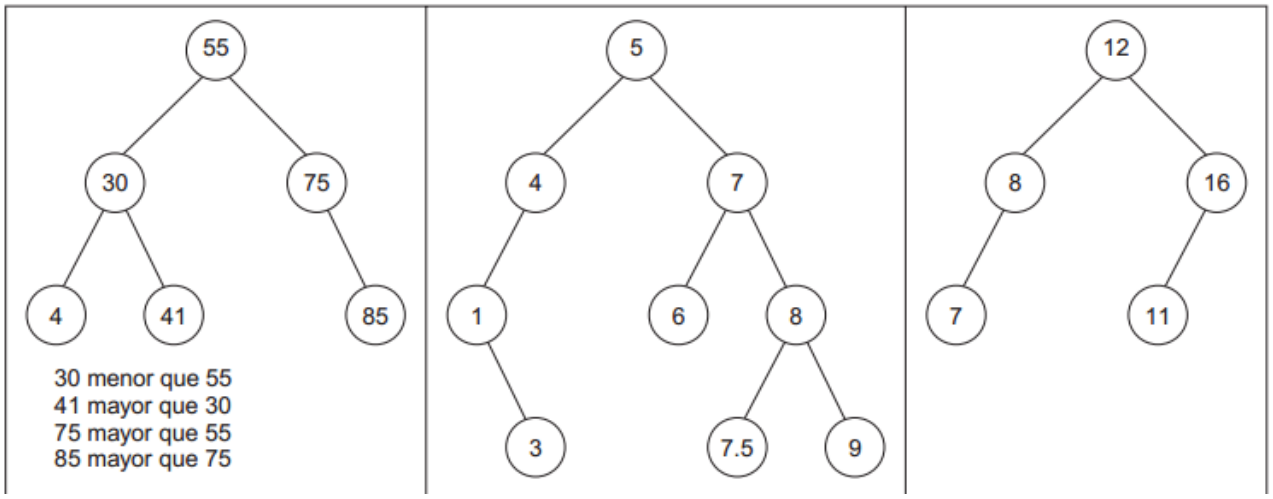
- **Preorden RID:** Visitar la raíz, recorrer en preorden el subárbol izquierdo, recorrer en preorden el subárbol derecho.
- **Inorden IRD:** Recorrer inorden el subárbol izquierdo, visitar la raíz, recorrer inorden el subárbol derecho.
- **Postorden IDR:** Recorrer en postorden el subárbol izquierdo, recorrer en postorden el subárbol derecho, visitar la raíz.

Existen otros tres recorridos más en profundidad pero apenas se usan. **RDI, DRI, DIR**.



2.3. Árbol binario de búsqueda

Un árbol binario de búsqueda es aquel que cualquier valor inferior al de la raíz se ubican en los hijos izquierdos, y los valores mayores a la raíz en los hijos derechos.



2.4. Operaciones en los árboles binarios de búsqueda

Las operaciones más usuales sobre árboles binarios de búsqueda son:

- búsqueda de un nodo
- inserción de un nodo
- borrado de un nodo.

2.4.1. Búsqueda

La búsqueda de un nodo comienza en el nodo raíz y sigue estos pasos:

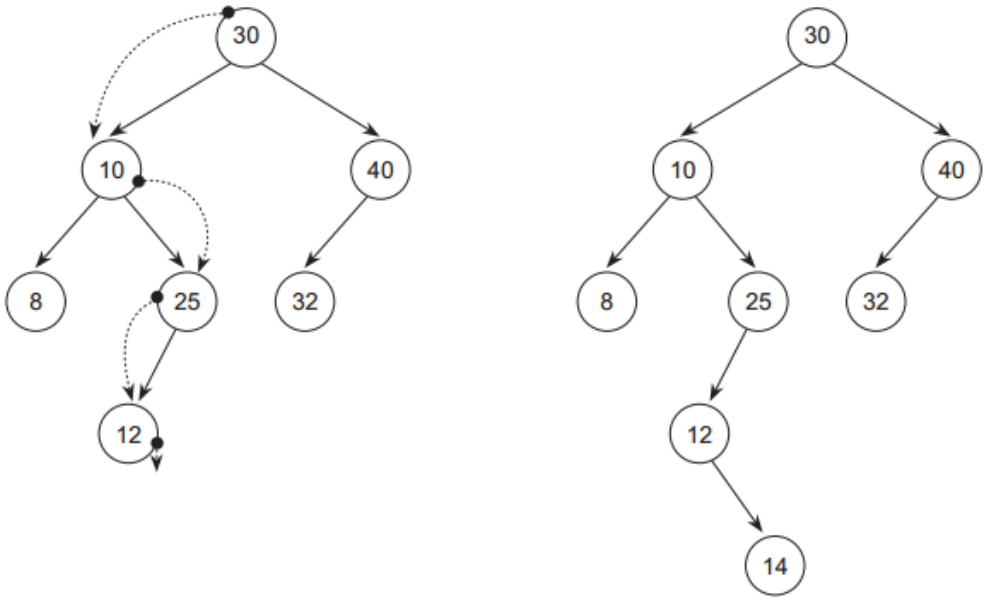
- Si el árbol está vacío la búsqueda termina con fallo.
- La clave buscada se compara con la clave del nodo raíz.
- Si las claves son iguales, la búsqueda se detiene con éxito.
- Si la clave buscada es mayor que la clave raíz, la búsqueda se reanuda en el subárbol derecho. Si la clave buscada es menor que la clave raíz, la búsqueda se reanuda con el subárbol izquierdo.

2.4.2. Inserción

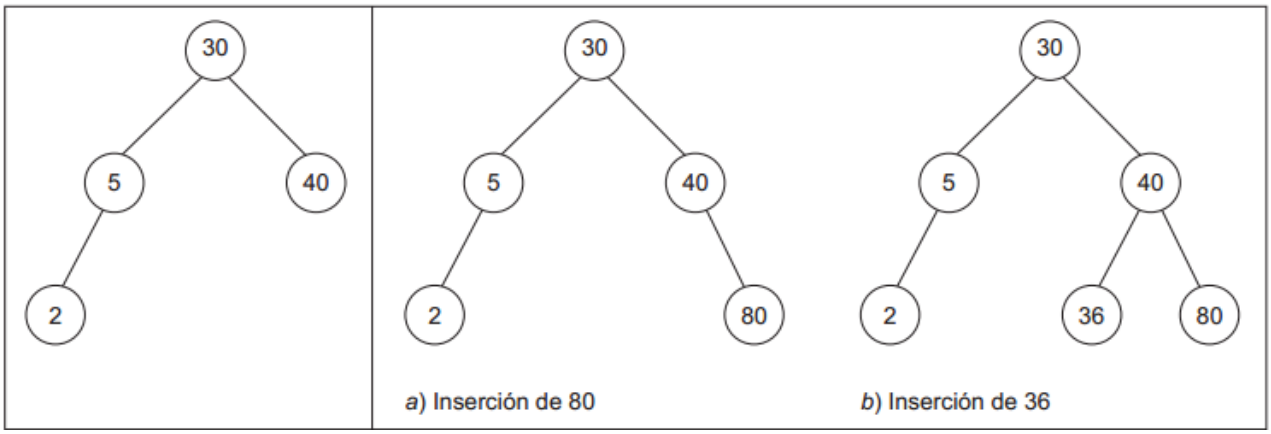
La operación de inserción de un nodo es una extensión de la operación de búsqueda. El algoritmo es:

- Asignar memoria para una nueva estructura nodo.
- Buscar en el árbol para encontrar la posición de inserción del nuevo nodo, que se colocará siempre como un nuevo nodo hoja.
- Enlazar el nuevo nodo al árbol. Para ello en el proceso de búsqueda hay que quedarse con el puntero que apunta a su padre y enlazar el nuevo nodo a su padre convenientemente. En caso de que no tenga padre (árbol vacío), se pone el árbol apuntando al nuevo nodo.

EJEMPLO 1. Inserción de 14 en el árbol de búsqueda expresado. El siguiente ejemplo muestra el camino a seguir para insertar el elemento de clave 14 en el árbol binario de búsqueda a continuación representado.



Inserción de los nodos 80 y 36

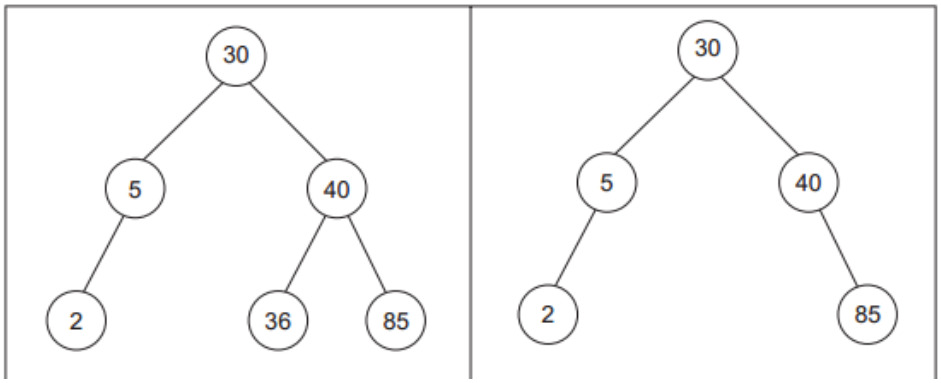


2.4.3. Borrado

La operación de borrado de un nodo es una extensión de la operación de búsqueda. Después de haber buscado el nodo a borrar hay que tener en cuenta:

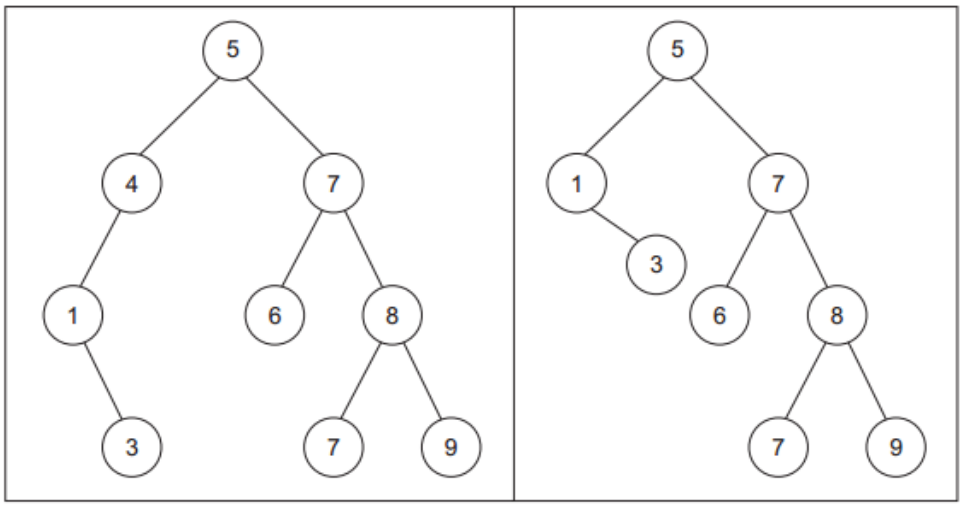
- Si el nodo es hoja, se suprime, asignando nulo al puntero de su antecesor.
- Si el nodo tiene único hijo. El nodo anterior se enlaza con el hijo del que se quiere borrar .
- Si tiene dos hijos. Se sustituye el valor almacenado en el nodo por el valor, inmediato superior (o inmediato inferior).
  Este nodo se encuentra en un avance a la derecha (izquierda) del nodo a borrar y todo a la izquierda (derecha), hasta que se encuentre NULL. Posteriormente, se borra el nodo que almacena el valor inmediato superior (o inmediato inferior) que tiene como máximo un hijo.
- Por último, hay que liberar el espacio en memoria ocupado el nodo.

Borrado de una hoja. Se borra la clave 36.



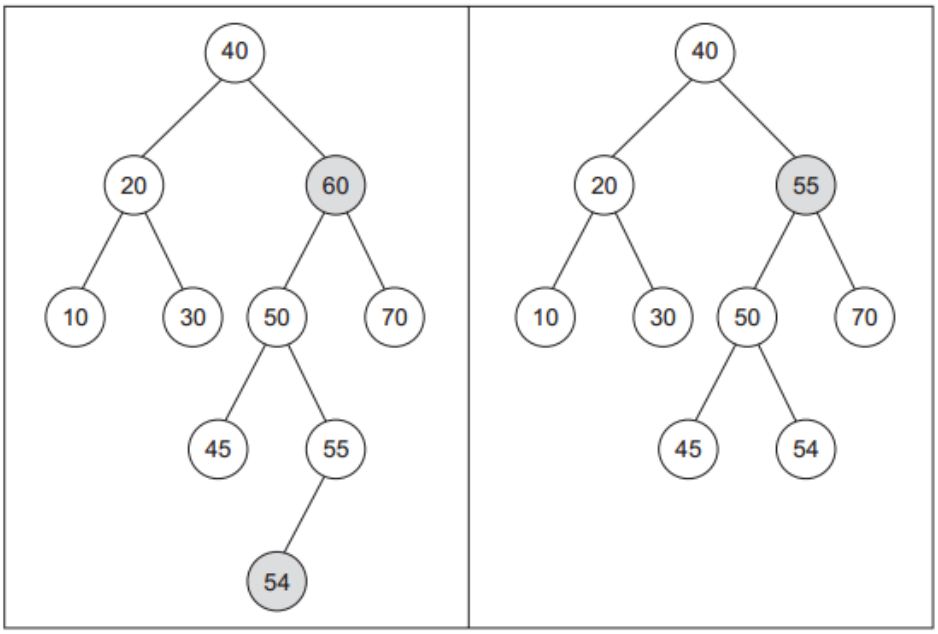
Borrado de un nodo con un solo hijo. Se borra la clave 4.

Se borra el nodo que contiene 5 puentes al nodo que contiene el 4 y se elimina el 4.

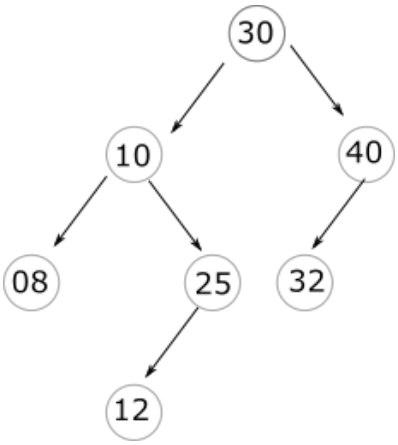


Borrado de un nodo con dos hijos. Se borr a la clave 60.

Se reemplaza 60 bien con el elemento mayor (55) en su subárbol izquierdo o el elemento más pequeño (70) en su subárbol derecho. Si se opta por reemplazar con el elemento mayor del subárbol izquierdo. Se mueve el 55 al raíz del subárbol y se reajusta el árbol.



EJEMPLO 2: Se desea escribir un programa en C que construya el siguiente árbol de búsqueda.



El programa deberá:

- Definir la estructura.
- Presentar un menú de opciones
- Construir el Árbol Binario.
- Agregar y borrar nodos.
- Recorrer el árbol.
- Buscar un elemento.

```
/* Arbol binario de bsqueda para la cdra informca 2*/
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

// Estructura
typedef struct nodo
{
    int dato;               // Valor del nodo
    struct nodo *ptrIzq;    // Puntero a la rama izquierda
    struct nodo *ptrDer;    // Puntero a la rama derecha
}Nodo;

Nodo nodoArbol; // estructura nodo
typedef Nodo *ptrNodoArbol; //

/*************************************************************
Declaracie prototipos de funci
*************************************************************/
int menu();

void insertarNodo(ptrNodoArbol *ptrArbol, int n);
void mostrarArbol(ptrNodoArbol ptrArbol, int);
void inOrden(ptrNodoArbol ptrArbol);
void preOrden(ptrNodoArbol ptrArbol);
void postOrden(ptrNodoArbol ptrArbol);
int buscarNodo(ptrNodoArbol ptrArbol, int);
void borrarNodo(ptrNodoArbol *ptrArbol, int );

/*************************************************************
Funcirincipal
*************************************************************/
void main()
{
    int n, opcion, nodoB, encontrarNodo, contador=0;
    char orden;
    Nodo *ptrRaiz = NULL; // Arbol Vac

    do
    {
        opcion = menu();
        switch(opcion)
        {
            case 1:
                printf("  Digite el valor del nodo a ingresar:\n");
                scanf("%d", &n);
                insertarNodo(&ptrRaiz, n);
                printf("\n");
                break;
            case 2:
                break;
            case 3:
                mostrarArbol(ptrRaiz, contador);
```

```
                break;

            case 4:
                printf("\n\t\t\t Ingrese el orden en que desea ver los datos.");
                printf("\n\t\t\t a. preorden.");
                printf("\n\t\t\t b. inorden.");
                printf("\n\t\t\t c. postorden.");
                printf("\n\n\t\t\t Orden: ");
                scanf("%s", &orden);
                if(orden == 'a')
                    preOrden(ptrRaiz);
                else if(orden == 'b')
                    inOrden(ptrRaiz);
                else if (orden == 'c')
                    postOrden(ptrRaiz);
                else
                    printf("\n\t Orden incorrecto.\n");
                break;
            case 5:
                printf("\n\t\t Ingrese el valor del nodo a buscar: ");
                scanf("%d", &nodoB);
                encontrarNodo= buscarNodo(ptrRaiz, nodoB);
                if(encontrarNodo)
                    printf("\n\n\t\t Nodo encontrado.");
                else
                    printf("\n\n\t\t :-( Nodo inexistente.");
            default: break;

    }


    }while(opcion!= 6);

    printf("\nPresione una tecla para finalizar.");
    getch();
}

//*****************************************************************************
int menu(ptrNodoArbol *ptrArbol)
{
    int opcion;

    printf("\n\t\t Menu:\n");
    printf("  1. Ingresar un nuevo nodo.\n");
    printf("  2. Borrar un nodo.\n");
    printf("  3. Mostrar arbol.\n");
    printf("  4. Recorrer arbol.\n");
    printf("  5. Buscar un nodo.\n");
    printf("  6. Salir.\n");
    printf("\n  Opcion: ");
    scanf("%d", &opcion);
    return opcion;
}
```

```
/*************************************************************
 Funciue agrega un nodo.
Comienza verificando que sea el nodo rael cual se debe crear
*************************************************************/
void insertarNodo(ptrNodoArbol *ptrArbol, int n)
{

    if(*ptrArbol == NULL)  // Si el arbol estaccrea el nodo ra
    {
        *ptrArbol = malloc(sizeof(nodoArbol)); // reserva memoria para el nodo ra
        if (*ptrArbol != NULL)  // si ya se reservmoria para la estructura
        {
            (*ptrArbol)->dato = n;  // carga la ra
            (*ptrArbol)->ptrIzq = NULL; // Inicialmente no tiene hijos
            (*ptrArbol)->ptrDer = NULL; // Si es una hoja tampoco tendrijos

        }
        else
        {
            printf("\nNo se reservo memoria para el nodo.");
            printf("\nPuede que sea insuficiente la memoria disponible ");
        }
    }
    else  // El ol ya existe
    {
        if(n < (*ptrArbol)->dato)  // Si el hijo es menor a la rava a la izquierda
            {
                insertarNodo(&( (*ptrArbol) -> ptrIzq), n); // Se llama recursivamente hasta e
            }
        else if (n > (*ptrArbol)->dato) // si el hijo es mayor a la ra
            {
                insertarNodo(&((*ptrArbol)->ptrDer), n); //  Se llama recursivamente para crea
            }
        else
            {
            printf("\t :-(  El valor ingresado ya existe. :-(");
            printf("\t Pruebe ingresando un valor diferente");
            }

    }

}// Fin de la funcion insertarNodo

/*****************************************************************************
Funciones que despliegan el ol binario de bsqueda
*****************************************************************************/
void mostrarArbol(ptrNodoArbol ptrArbol, int contador)
{
    int i;
    if(ptrArbol == NULL)
    {
//        printf("\n\t\t LLego a la ultima hoja \n\n");
```

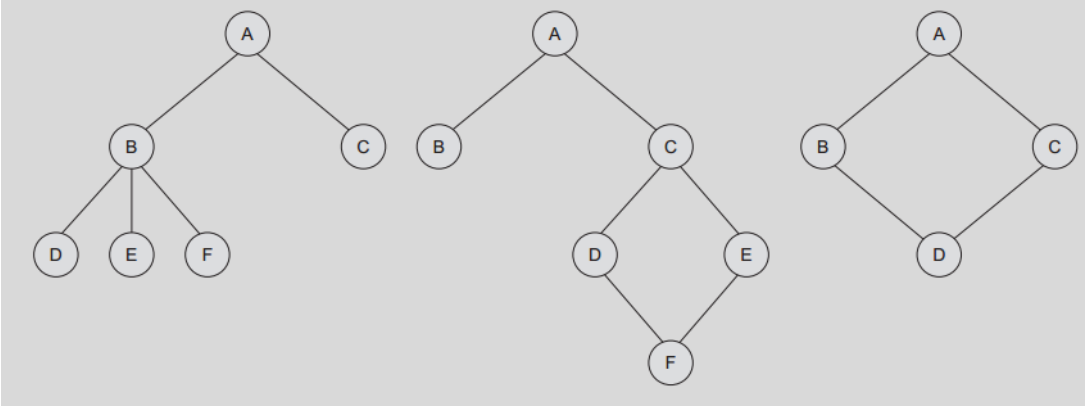
```
        return;
    }
    else
    {
        mostrarArbol(ptrArbol->ptrDer, contador+1);  // recorre el ol en forma recursiva
        for(i=0; i<contador; i++)
        {
            printf("\t");
        }
        printf("%d\n", ptrArbol->dato);
        mostrarArbol(ptrArbol->ptrIzq, contador+1);
    }
}



void inOrden(ptrNodoArbol ptrArbol)
{
   if(ptrArbol != NULL)  // Si existe un arbol
    {
        inOrden(ptrArbol->ptrIzq);           // recorre primero recursivamente el lado izquier
        printf("%3d", ptrArbol->dato);      // imprime primero los valores menores a la raiz
        inOrden(ptrArbol->ptrDer);          // imprime luego los valores mayores a la raiz
    }
}


void preOrden(ptrNodoArbol ptrArbol)
{
   if(ptrArbol != NULL)
    {
        printf("%3d", ptrArbol->dato);
        preOrden(ptrArbol->ptrIzq);
        preOrden(ptrArbol->ptrDer);
    }
}

void postOrden(ptrNodoArbol ptrArbol)
{
   if(ptrArbol != NULL)
    {

        preOrden(ptrArbol->ptrIzq);
        preOrden(ptrArbol->ptrDer);
        printf("%3d", ptrArbol->dato);
    }
}


/**************************************************************************
Funciue busca un nodo
**************************************************************************/
int buscarNodo(ptrNodoArbol ptrArbol, int nodoB)
```

```
{
    if(ptrArbol == NULL)
        return 0;
    else if(nodoB < (ptrArbol->dato))
    {
        buscarNodo((ptrArbol->ptrIzq), nodoB);
    }
    else if(nodoB > (ptrArbol->dato))
    {
        buscarNodo((ptrArbol->ptrDer), nodoB);
    }
    else
        return 1;
}
```

3. Problemas

P1. Modificar el programa del ejemplo 2 para incluir la opción de borrar un nodo

P2. Explicar porqué cada una de las siguientes estructur as no es un árbol binario.



P3. Para cada una de las siguientes listas de letras:

a Dibujar el árbol binario de búsqueda que se construye cuando las letras se insertan en el orden dado.

b Realizar recorridos inorden, preorden y postorden del árbol y mostrar la secuencia de letras que resultan en cada caso.

(I) M,Y,T,E,R **(II)** R,E,M,Y,T **(III)** T,Y,M,E,R **(IV)** C,O,R,N,F,L,A,K,E,S