Árboles binarios

Taller de programación II
Analista Universitario en Sistemas
Dto. de Informática
Instituto Politécnico Superior

Definición de Árbol

- Un árbol es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos.

Repaso de conceptos

En relación con otros nodos:

- **Nodo hijo:** cualquiera de los nodos apuntados por uno de los nodos del árbol. En el ejemplo, 'L' y 'M' son hijos de 'G'.
- **Nodo padre:** nodo que contiene un puntero al nodo actual. En el ejemplo, el nodo 'A' es padre de 'B', 'C' y 'D'.
- Los árboles con los que trabajaremos tienen otra característica importante: cada nodo sólo puede ser apuntado por otro nodo, es decir, cada nodo sólo tendrá un padre. Esto hace que estos árboles estén fuertemente jerarquizados, y es lo que en realidad les da la apariencia de árboles.

Conceptos, continuación:

En cuanto a la posición dentro del árbol:

- **Nodo raíz:** nodo que no tiene padre. Este es el nodo que usaremos para referirnos al árbol. En el ejemplo, ese nodo es el 'A'.
- **Nodo hoja:** nodo que no tiene hijos. En el ejemplo hay varios: 'F', 'H', 'I', 'K', 'L', 'M', 'N' y 'O'.
- **Nodo rama:** Estos son los nodos que no pertenecen a ninguna de las dos categorías anteriores. En el ejemplo: 'B', 'C', 'D', 'E', 'G' y 'J'.

Características

- Todos los nodos contienen el mismo número de punteros, es decir, usaremos la misma estructura para todos los nodos del árbol. Esto hace que la estructura sea más sencilla, y por lo tanto también los programas para trabajar con ellos.
- Tampoco es necesario que todos los nodos hijos de un nodo concreto existan. Es decir, que pueden usarse todos, algunos o ninguno de los punteros de cada nodo.
- Un árbol en el que en cada nodo o bien todos o ninguno de los hijos existe, se llama **árbol completo**.
- Dado un nodo cualquiera de la estructura, podemos considerarlo como una estructura independiente. Es decir, un nodo cualquiera puede ser considerado como la raíz de un árbol completo.

Según el tamaño

Orden: es el número potencial de hijos que puede tener cada elemento de árbol. De este modo, diremos que un árbol en el que cada nodo puede apuntar a otros dos es de *orden dos*, si puede apuntar a tres será de *orden tres*, etc.

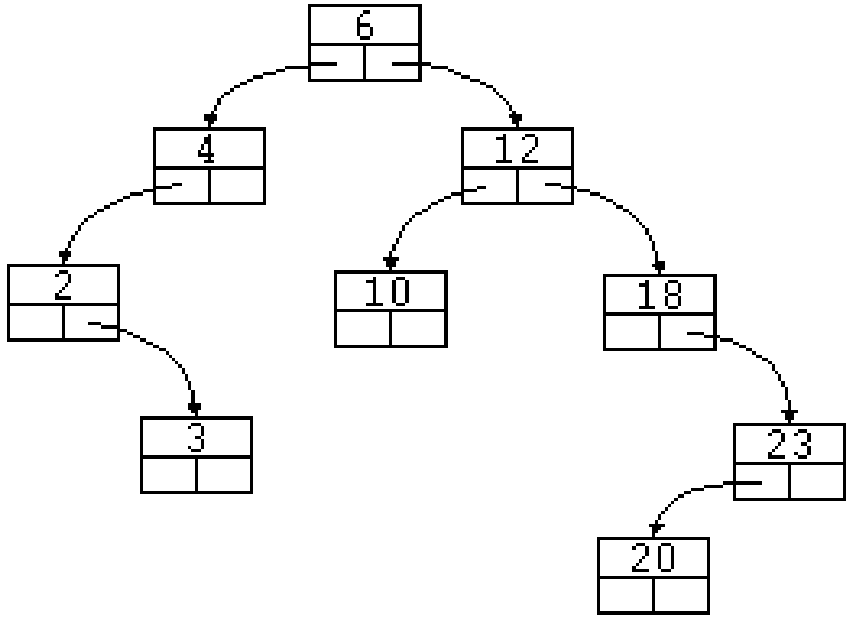
Grado: el número de hijos que tiene el elemento con más hijos dentro del árbol. En el árbol del ejemplo, el grado es tres, ya que tanto 'A' como 'D' tienen tres hijos, y no existen elementos con más de tres hijos.

Nivel: se define para cada elemento del árbol como la distancia a la raíz, medida en nodos. El nivel de la raíz es cero y el de sus hijos uno. Así sucesivamente. En el ejemplo, el nodo 'D' tiene nivel 1, el nodo 'G' tiene nivel 2, y el nodo 'N', nivel 3.

Altura: la altura de un árbol se define como el nivel del nodo de mayor nivel. Como cada nodo de un árbol puede considerarse a su vez como la raíz de un árbol, también podemos hablar de altura de ramas. El árbol del ejemplo tiene altura 3, la rama 'B' tiene altura 2, la rama 'G' tiene altura 1, la 'H' cero, etc.

Árboles binarios

Se trata de árboles de orden 2 en los que se cumple que para cada nodo, el valor de la clave de la raíz del subárbol izquierdo es menor que el valor de la clave del nodo y que el valor de la clave raíz del subárbol derecho es mayor que el valor de la clave del nodo.



Árboles binarios

Tenemos 3 métodos fundamentales:

- Insertar

- Eliminar

- Es Miembro

Teniendo en cuenta el concepto anterior, la búsqueda se simplifica por la estructura ordenada que tiene el árbol. En caso que el dato no este en el nodo, busco a la derecha o a la izquierda según sea el caso. Y repito el procedimiento.

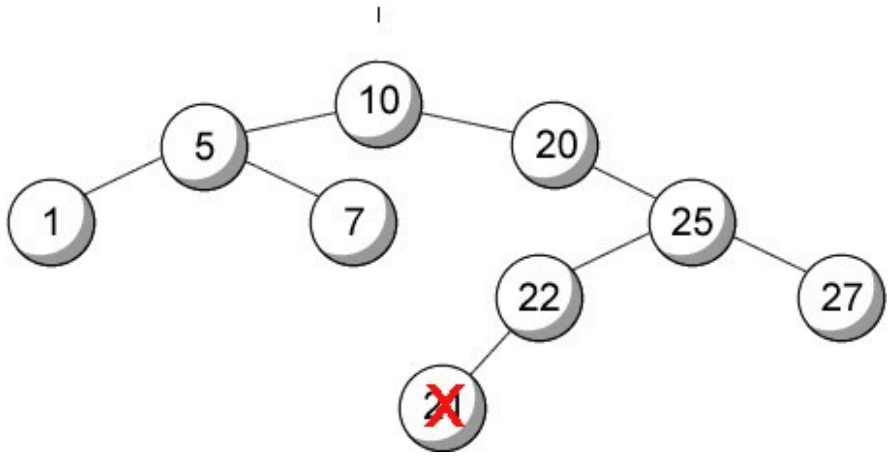
De modo similar, el insertar también es simple. El dato se guardará en un nodo que se ira ubicando en el árbol en la posición izquierda o derecha dependiendo al valor encontrado en el nodo existente.

Árbol binario

Eliminar

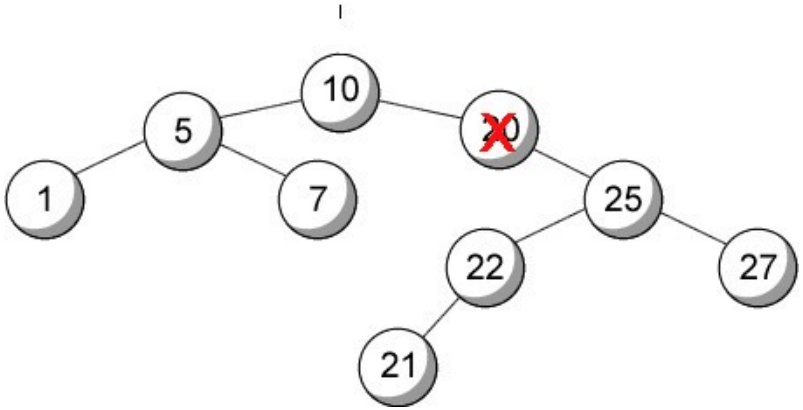
Para eliminar, vemos que tenemos dos prototipos y ambos son necesarios.
Analizaremos las situaciones:

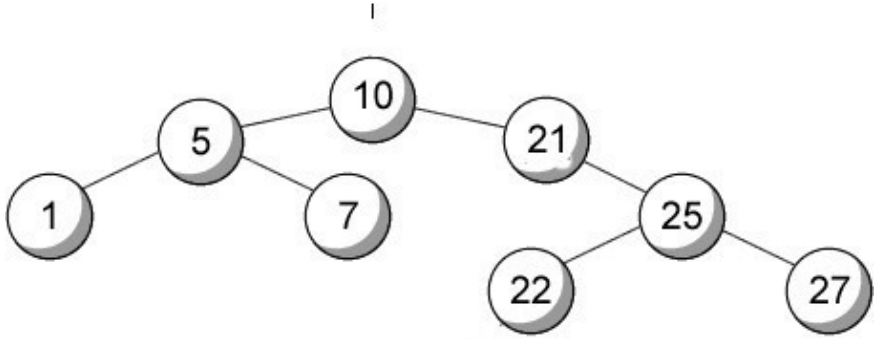
1- Hay que encontrar el nodo a eliminar
2- Si es un nodo terminal (sin nodos dependientes) lo borramos directamente



Árbol binario

3- Si quisiéramos borrar un nodo con hijos, lo importante es no dejar al árbol inconexo... Para eso reemplazaremos el dato a borrar con el valor mas pequeño de la rama derecha del nodo a eliminar.

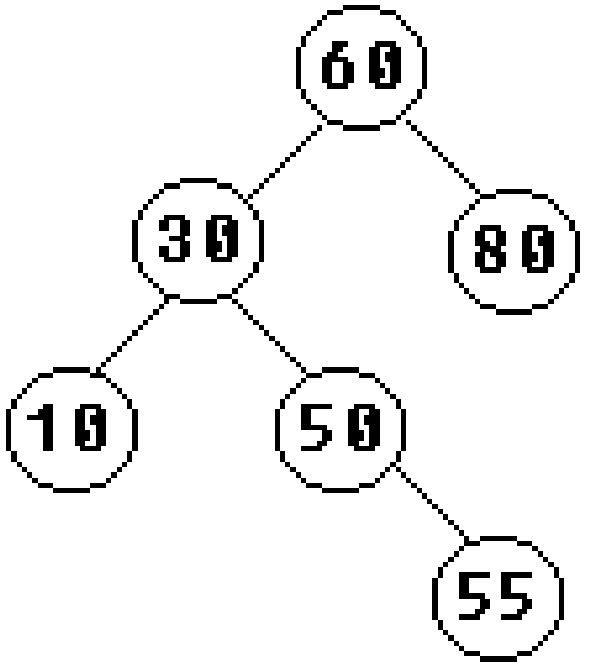




Para esta acción usaremos como función auxiliar, la suprime minimo que veremos en el código.

Definición de Recorridos

- **inorden** : recorrer en inorden el subárbol izquierdo , visitar el elemento de la raíz y luego recorrer en inorden el subárbol derecho.
- **preorden** : visitar el elemento de la raíz , recorrer en preorden el subárbol izquierdo y luego recorrer en preorden el subárbol derecho.
- **postorden** : recorrer en postorden el s luego recorrer en postorden el subárb finalmente visitar el elemento de la raí



- Ejemplo de Recorridos

  inorden : 10 , 30 , 50 , 55 , 60 , 80

  preorden : 60 , 30 , 10 , 50 , 55 , 80

  postorden : 10 , 55 , 50 , 30 , 80 , 60

void inserta (arbol**, int)

```
void inserta ( arbol ** A , int x )
{
    if (* A == NULL ) {
        * A = ( arbol *) malloc ( sizeof
( arbol ) ) ;
        (* A ) -> dato = x ;
        (* A ) -> h_izq = NULL ;
        (* A ) -> h_der = NULL ;
    }
    else {
        if ( x < (* A ) -> dato )
            inserta (&((* A ) -> h_izq ) , x ) ;
        else if ( x > (* A ) -> dato )
            inserta (&((* A ) -> h_der ) , x ) ;
    }
}
```

int es_miembro (arbol*, int)

```
int es_miembro ( arbol * A , int x ) {
    if ( A == NULL )
        return -1;
    else if (A - > dato == x )
        return 1;
    else if (A - > dato > x )
        return es_miembro (A - > h_izq , x ) ;
    else
        return es_miembro (A - > h_der , x ) ;
}
```

int suprime_min (arbol**)

```
int suprime_min ( arbol ** A )
{
    int v_ref ;
    if ((* A ) -> h_izq == NULL ) {
        v_ref = (* A ) -> dato ;
        arbol * tmp = * A ;
        *A = (*A ) -> h_der ;
        free ( tmp ) ;
        return v_ref ;
    } else {
        return suprime_min (&((* A ) -> h_izq ) ) ;
    }
}
```

void suprime (arbol**, int)

```
void suprime ( arbol ** A , int x ) {
    if (* A != NULL ) {
        if ( x < (* A ) -> dato )
            suprime (&((*A )->h_izq ) , x ) ;
        else if ( x > (*A )->dato )
            suprime (&((*A )->h_der ) , x ) ;
    // Lo encontre
        else if ((*A )->h_izq == NULL && (*A)->h_der == NULL ) {
            arbol* tmp = * A ;
            *A = NULL ;
            free ( tmp ) ;
        }
```

Suprime cont...

```
else if ((*A)->h_izq == NULL ) {
            arbol* tmp = * A ;
            *A = (*A)->h_der ;
            free ( tmp ) ;
        } else if ((*A)->h_der == NULL ) {
            arbol* tmp = *A ;
            *A = (*A)->h_izq ;
            free ( tmp ) ;
        } else { // ambos hijos estan presentes
            (*A)->dato = suprime_min (&((*A)->h_der)) ;
        }
    }
}
```