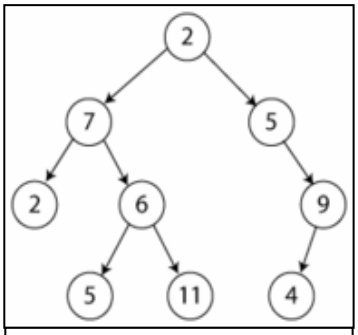
Definiciones Varias:



Un árbol binario sencillo de tamaño 9 y altura 3, con un nodo raíz cuyo valor es 2.

1.- En ciencias de la computación, un árbol binario es una estructura de datos en la cual cada nodo siempre tiene un hijo izquierdo y un hijo derecho. No puede tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado una hoja. En el caso contrario el hijo es llamado un nodo interno.

2.- En teoría de grafos: «Un árbol binario es un grafo conexo, acíclico y no dirigido tal que el grado de cada vértice no es mayor a 3». De esta forma sólo existe un camino entre un par de nodos.

Un árbol binario con enraizado es como un grafo que tiene uno de sus vértices, llamado *raíz*, de grado no mayor a 2. Con la raíz escogida, cada vértice tendrá un único padre, y nunca más de dos hijos.

Un árbol binario es un árbol en el que ningún nodo puede tener más de dos subárboles. En un árbol binario cada nodo puede tener cero, uno o dos hijos (subárboles). Se conoce el nodo de la izquierda como hijo izquierdo y el nodo de la derecha como hijo derecho.

Características de los árboles binarios

- El número de hojas en un árbol binario completo es siempre igual al número de nodos internos más uno.
- El número de nodos de un árbol binario completo de altura h es igual a $\mathbf{2^{h+1} - 1}$. Como todo árbol binario completo es también lleno, $\mathbf{2^h}$ de esos nodos son hojas y $\mathbf{2^h - 1}$ son nodos internos.
- Un árbol binario de altura *h* se dice ***semicompleto*** si los nodos de los niveles *h* y *h-1* son los únicos de grado inferior a 2 y las hojas del último nivel ocupan las posiciones más a la izquierda del mismo.
- La aplicación más importante de los árboles es organizar la información de manera jerárquica para acelerar los procesos de búsqueda, inserción y borrado.
- Normalmente, la clave de búsqueda se extrae de la propia información, directamente o mediante transformaciones adecuadas.
- Para clasificar la información sin ambigüedad, es necesario establecer entre las claves de búsqueda un conjunto de condiciones mutuamente excluyentes, tal que una y sólo una de ellas sea cierta.

Implementación en C

Un árbol binario puede declararse de varias maneras. Algunas de ellas son:

```
struct nodoarbol{
        int clave;
        struct nodoarbol *izq;
        struct nodoarbol *der;
};
typedef struct nodoarbol NODO;
```

```
typedef struct tArbol{
  int clave;
  int hIzquierdo, hDerecho;
} tArbol;

tArbol árbol[NUMERO_DE_NODOS];
```

En el caso de un árbol binario casi-completo (o un árbol completo), puede utilizarse un sencillo arreglo de enteros con tantas posiciones como nodos deba tener el árbol. La información de la ubicación del nodo en el árbol es implícita a cada posición del arreglo. Así, si un nodo está en la posición *i*, sus hijos se encuentran en las posiciones $2i+1$ y $2i+2$, mientras que su padre (si tiene), se encuentra en la posición truncamiento($(i-1)/2$) (suponiendo que la raíz está en la posición cero). Este método se beneficia de un almacenamiento más

compacto y una mejor localidad de referencia, particularmente durante un recorrido en preorden. La estructura para este caso sería por tanto:

int árbol[NUMERO_DE_NODOS];

Para la creación de un árbol binario dinámico, podría usarse una función como la siguiente:

```
NODO *creaArbol(NODO *r)
{
char c;
r= new(NODO);
printf("\n\t Ingrese el valor del nodo:"); scanf("%d",&r->clave); getchar();
printf("El nodo %d, ¿Tiene hijo izquierdo (s/n)?:",r->clave); scanf("%c",&c);getchar();
if (c=='s') r->izq=creaArbol(r->izq);
else { r->izq = NULL;}
printf("El nodo %d, ¿Tiene hijo derecho (s/n)?:",r->clave); scanf("%c",&c);getchar();
if (c=='s')r->der = creaArbol(r->der);
else {r->der=NULL;}
return r;
}
```

Recorridos en Amplitud y Profundidad (Preorden, Inorden, Postorden) sobre árboles binarios

Recorrido en preorden: En este tipo de recorrido se realiza una acción sobre el nodo actual y posteriormente se trata el subárbol izquierdo en Preorden, y cuando se haya concluido, el subárbol derecho en Preorden. En el árbol de la figura 1 el recorrido en preorden sería: 2, 7, 2, 6, 5, 11, 5, 9 y 4.

```
void preorden(tArbol *a){
 if (a != NULL) {
   visita(a);     //Manipula al nodo
   preorden(a->hIzquierdo);
   preorden(a->hDerecho);
 }
}
```

Implementación en pseudocódigo de forma iterativa, se considera una pila inicialmente vacía:

```
push(s, NULL);      //inicializa la pila
push(s, raíz);      //inserta la raíz
while (s != NULL){
   p = pop(s);   //saca un elemento de la pila
   tratar(p);   //realiza operaciones sobre el nodo p
   if (p->hder != NULL)  push(s, p->hder);
   if (p->hizq != NULL   push(s, p->hizq); }
```

Recorrido en postorden: En este caso se trata primero el subárbol izquierdo, después el derecho y por último el nodo actual. En el árbol de la figura el recorrido en postorden sería: 2, 5, 11, 6, 7, 4, 9, 5 y 2.

```
void postorden(tArbol *a) {
 if (a != NULL) {
   postorden(a-<hIzquiedo);
   postorden(a->hDerecho);
   visita(a);     //Manipula al nodo
 }
}
```

Implementación en pseudocódigo de forma iterativa, se considera una pila inicialmente vacía:

¡ Desarrollar!

Recorrido en inorden: En este caso se trata primero el subárbol izquierdo, después el nodo actual y por último el subárbol derecho. En un ABB este recorrido daría los valores de clave ordenados de menor a mayor. En el árbol de la figura el recorrido en inorden sería: 2, 7, 5, 6, 11, 2, 5, 4 y 9.

void inorden(tArbol *a)
{
  if (a != NULL) {
  inorden(a-<hIzquierdo);
  visita(a); //manipula al nodo
  inorden(a-<=hDerecho);
  }
}
	Implementación en pseudocódigo de forma iterativa, se considera una pila inicialmente vacía:
	
	¡ Desarrollar!

Recorridos en amplitud (o por niveles)

En este caso el recorrido se realiza en orden por los distintos niveles del árbol. Así, se comenzaría tratando el nivel 1, que sólo contiene el nodo raíz, seguidamente el nivel 2, el 3 y así sucesivamente. En el árbol de la figura el recorrido en amplitud sería: 2, 7, 5, 2, 6, 9, 5, 11 y 4.

Al contrario que en los métodos de recorrido en profundidad, el recorrido por niveles no es de naturaleza recursiva. Por ello, se debe utilizar una fila para recordar los subárboles izquierdos y derecho de cada nodo.