Contenido

PARTE II: ESTRUCTURAS DE DATOS AVANZADAS

TEMA 4. - La Estructura de datos Árbol

4.1 Árboles, definiciones

Conceptos básicos

Los árboles son una de las estructuras de datos no lineales que sirve para representar estructuras de información jerárquicas y direcciones o etiquetas de una manera organizada.

Ejemplos comunes de árboles:

- La organización de una empresa (organigrama).
- Árbol genealógico de una persona
- Organización de torneos deportivos (tabla de partidos)

Ejemplos de aplicaciones computacionales de los árboles:

- Realizar ordenaciones de datos.
- Realizar búsquedas.
- Asignar bloques de memoria de tamaño variable
- Organizar tablas de símbolos en compiladores
- Representar tablas de decisión
- Solucionar juegos
- Probar teoremas

Definición de árbol

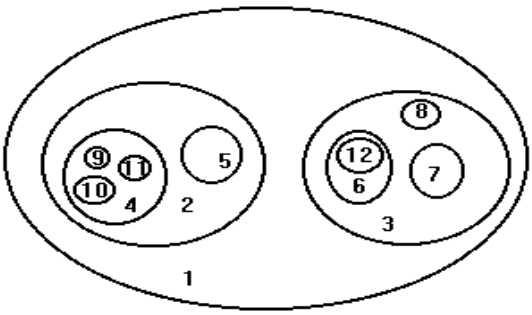
Un árbol es un conjunto finito T de uno o más nodos tal que:

a) Existe un nodo especial llamado la *raíz* de árbol
b) Los nodos restantes están particionados en m $\geq$ 0 conjuntos disjuntos $T_1,...,T_m$ y cada uno de estos conjuntos es a su vez un árbol. Los árboles $T_1,...,T_m$ son llamados *subárboles* de la raíz

Cualquier nodo es la raíz de un subárbol que consiste de él y los nodos debajo de él. Por ello se dice que la definición de árbol es recursiva.

Representaciones gráficas de un árbol

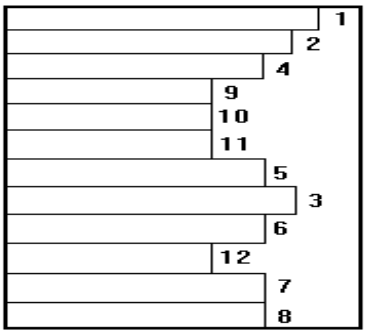
1 Representación con conjuntos anidados:



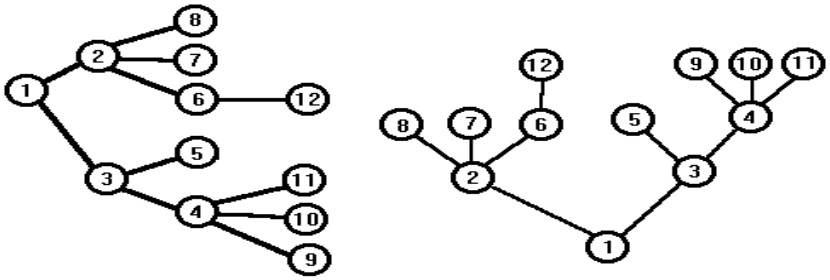
2 Representación por paréntesis anidados.

(1(2(4(9,10,11),5),3(6(12),7,8)))

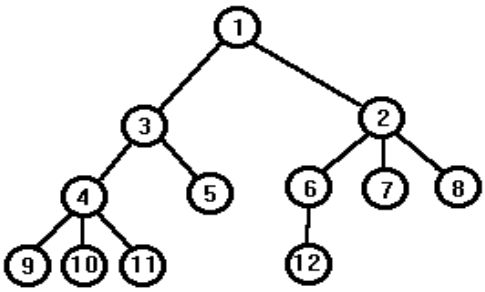
3 Representación por indentación



- Representación con grafos



La forma más común para representar un árbol es usando un grafo.



Terminología

1 Dado un conjunto de *E* elementos:

Un árbol puede estar vacío.

Un árbol no vacío puede constar de un solo elemento llamado *nodo raíz*.

Un árbol consta de un nodo raíz *e* □ *E*, conectado por arcos directos a un número finito de otros árboles.

2 Definiciones sobre Árboles

Un *vértice o nodo* del árbol puede tener un nombre y puede tener asociada a él una información.

El primer nodo de un árbol es llamado *raíz*.

Las flechas que conectan un nodo a otro se llaman arcos o ramas.

Los nodos terminales son aquellos que no conducen a algún nodo, se llaman *hojas*.

Los nodos que no son hojas se llaman *nodos no terminales* o *nodos internos.*

Si un nodo n_1 tiene una rama hacia el nodo n_2 , se dice que n_2 es un *nodo hijo* de n_1.

Un *camino* es una lista de ramas contiguas que van de n_1 a n_2. Una propiedad que define a los árboles es que existe exactamente un camino entre la raíz y cada uno de los otros nodos en el árbol.

La *longitud de un camino* es el número de nodos del camino menos uno o de forma equivalente es el número de arcos que contiene el camino Por tanto, hay un camino de longitud cero de cualquier nodo a si mismo.

El número de hijos que parten de un nodo es llamado el *grado* del nodo. El grado de un árbol es el grado máximo de los nodos del árbol.

El *nivel* de un nodo es la longitud del camino que lo conecta al nodo raíz.

La *profundidad* del nodo n_k es el largo del camino entre la raíz del árbol y el nodo n_k. La profundidad de la raíz es siempre 0.

La *altura* de un nodo n_k es el máximo largo de camino desde n_k hasta alguna hoja. Esto implica que la altura de toda hoja es 0. La altura de un árbol es igual a la altura de la raíz y tiene el mismo valor que la profundidad de la hoja más profunda. La altura de un árbol vacío es -1.

Un *subárbol* de un árbol es un subconjunto de nodos del árbol conectados pro ramas del propio árbol, esto es, a su vez un árbol.

Ejemplo:
Considerar el siguiente árbol:

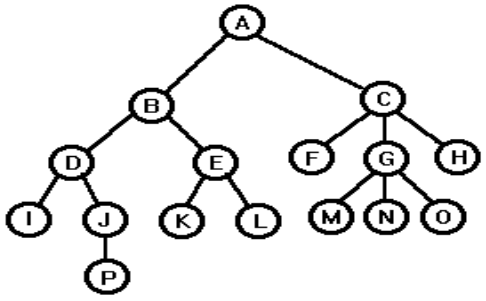


Figura 5. Ejemplo de un árbol

Nodo raíz: A
Nodos no terminales: A, B, C, D, E, G, J
Nodos padres A cuyos hijos son: A, C
B cuyos hijos son: D, E
C cuyos hijos son: F, G, H
D cuyos hijos son: I, J

E cuyos hijos son: K, L
G cuyos hijos son: M, N, O
J cuyos hijo es: P

Nodos hojas: I, P, K, L, F, M, N, O, H
Grados de los nodos:

Nodos de grado 0 son: *I, P, K, L, F, M, N, O, P* y *H* (son las hojas del árbol).
Nodo de grado 1 es *J*.
Nodos de grado 2 son *A, B, D* y *E*.
Nodos de grado 3 son *C* y *G*.
El grado del árbol es 3.

El camino del nodo *A* al nodo *P* es (*A, B, D, J, P*) y longitud de camino es 4.
Los niveles del árbol se presentan en la figura 6.

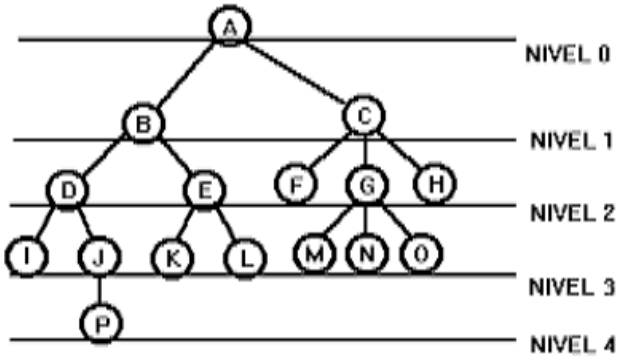


Figura 6. Niveles de un árbol

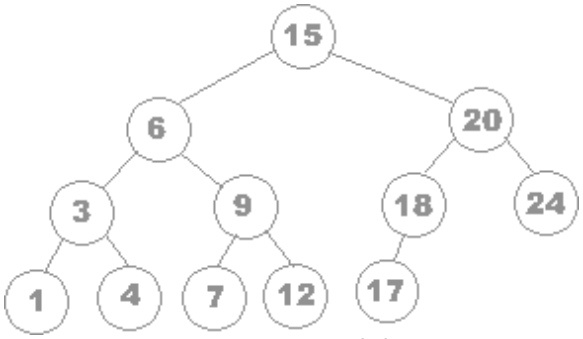
De lo anterior, se ve que el árbol es de altura 4.

(Un conjunto de árboles es llamado un *bosque.* Existe una pequeña diferencia entre un árbol y un bosque; si se elimina la raíz de un árbol se obtiene un bosque, y, recíprocamente, si se añade un nodo a un bosque se obtiene un árbol.)

4.2 Árboles binarios

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. El nodo de la izquierda representa al hijo izquierdo y el nodo de la derecha representa al hijo derecho.

Un árbol binario (AB) es una estructura recursiva porque cada nodo es la raíz de su propio subárbol y tiene hijos que son a su vez raíces de sus subárboles izquierdo y derecho.



Arbol A

Figura 7 Ejemplos de un árbol es binarios

Un **árbol** se llama **degenerado** cuando sólo contiene una hoja y cada nodo NO hoja sólo tiene un hijo. Este tipo de árbol equivale a una lista enlazada. Por ejemplo el árbol de la figura 7 (b).

Árbol equilibrado

Para determinar si un AB está equilibrado se calcula su factor de equilibrio (FE).

El FE es la diferencia entre la altura del subárbol derecho menos el izquierdo.

$$FE = h_{SD} - h_{SI}$$

De los árboles de la figura 7, su FE es

FE(a)=
FE(b)=
FE(c)=
FE(d)=

Un **árbol** está **perfectamente equilibrado** si su FE es 0, al igual que el de sus subárboles (caso raro). Un AB está equilibrado si la altura de sus subárboles difiere en no más de 1 (-1, 0, +1) y sus subárboles son también equilibrados.

AB completos

Un AB completo (ABC) de profundidad N es un árbol en el que para cada nivel, del 0 al nivel N-1 tiene un conjunto lleno de nodos y todos los nodos hoja a nivel N ocupan las posiciones más a la izquierda del árbol.

Árbol lleno

Es un árbol completo que contiene en su último nivel el número máximo de nodos.

Fórmula para **obtener la profundidad** de un árbol a partir del número de nodos (aplica a un AB completo).

$h = \text{entero}(\log_2 n) + 1$ donde 2^0 nodos en el nivel 0
2^1 nodos en el nivel 1
2^k-1 nodos en el nivel k.

Ejemplos:

Se tiene n=10000 elementos para formar un ABC, su profundidad será de ..

$= \text{entero}(\log_2 10000) + 1 = \text{entero}(13.28) + 1 = \underline{14}$

Un árbol lleno de profundidad 4 (niveles 0 a 3) tiene…….. 2^4-1 = <u>15 nodos</u>.

Representación ligada de árboles binarios

La representación más usada es la ligada o dinámica, donde cada nodo o celda tiene la siguiente forma:

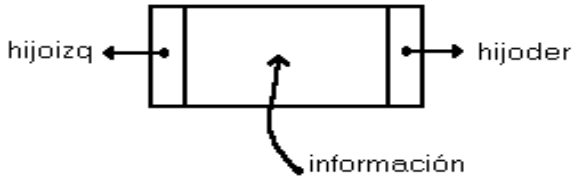


Figura 8 Representación ligada de un nodo para AB

Un nodo de un árbol puede representarse como:

```
typedef struct _nodo{
        int info;
        struct _nodo *izq;
        struct _nodo *der;
}tiponodo;
```

Y un árbol como:

```
typedef tiponodo *Arbol;
```

Ejemplo:

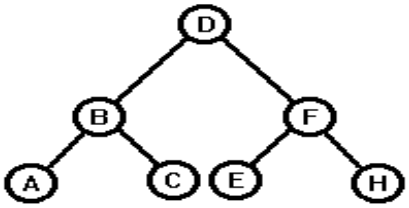
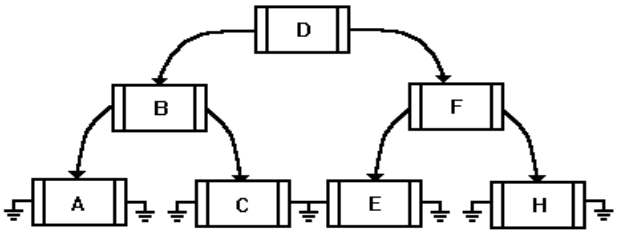


Figura 9 Ejemplo de la representación ligada de un AB

El árbol anterior, en representación ligada:



Tipos de AB: AB de expresiones y AB de búsqueda

Árboles de expresiones matemáticas

La figura 10 muestra un ejemplo de un *árbol de expresiones matemáticas*. En un árbol de expresiones las hojas corresponden a los *operandos* de la expresión (variables o constantes), mientras que los nodos restantes contienen *operadores*. Dado que los operadores matemáticos son binarios (o unarios como en el caso del operador signo -), un árbol de expresiones resulta ser un árbol binario.

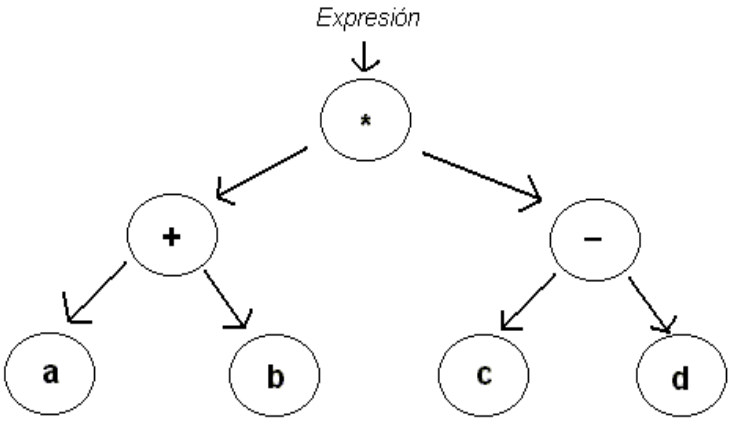


Figura 10 Ejemplo de un AB de expresión

Un árbol de expresiones se puede evaluar de la siguiente forma:

- Si la raíz del árbol es una constante o una variable se retorna el valor de ésta.
- Si la raíz resulta ser un operador, entonces recursivamente se evalúan los subárboles izquierdo y derecho, y se retorna el valor que resulta al operar los valores obtenidos de las evaluaciones de los subárboles con el operador respectivo.

Recorrido en árboles binarios

Para recorrer un árbol, existen varias formas de lograrlo, las 3 más comunes son preorden, inorden y postorden.

Para recorrer un árbol binario no vacío T en **preorden** o previo:

1. visitar la raíz
2. recorrer recursivamente el subárbol izquierdo
3. recorrer recursivamente el subárbol derecho

Para recorrer un árbol binario no vacío T en **inorden** o simétrico:

1. recorrer recursivamente el subárbol izquierdo
2. visitar la raíz
3. recorrer recursivamente el subárbol derecho

Para recorrer un árbol binario no vacio T en **postorden** o posterior:

1. recorrer recursivamente el subárbol izquierdo
2. recorrer recursivamente el subárbol derecho

3. visitar la raíz

Ejemplo:
Considerar el siguiente árbol:

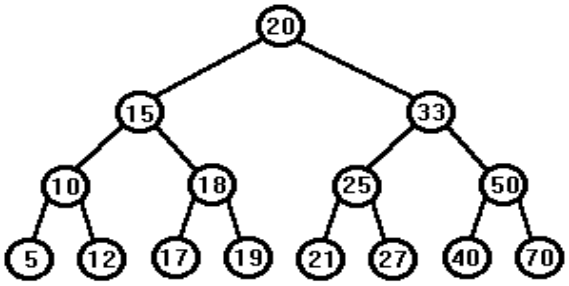


Figura 11 Ejemplo de un AB para obtener sus recorridos

El resultado que se obtiene al recorrer el árbol en

preorden: 20, 15 10, 5, 12, 18, 17, 19, 33, 25, 21, 27, 50, 40, 70
inorden: 5, 10, 12, 15, 17, 18, 19, 20, 21, 25, 27, 33, 40, 50, 70
postorden: 5, 12, 10, 17, 19, 18, 15, 21, 27, 25, 40, 70, 50, 33, 20

Árboles binarios de búsqueda

Dado un conjunto de nodos (con un campo asignado como llave de búsqueda) T es un árbol binario de búsqueda si:

1. T es vacío, o
2. T es de la forma

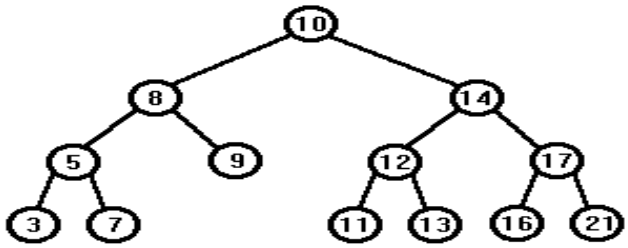
y
a) la llave de búsqueda de n es mayor que todas las llaves de búsqueda de T_I
b) la llave de búsqueda de n es menor que todas las llaves de búsqueda de T_D
c) T_I y T_D son árboles binarios de búsqueda

Búsqueda
Una ventaja de los árboles binarios de búsqueda es que el algoritmo de búsqueda surge de una manera natural. Si *x* es la llave de la raíz de un árbol binario de búsqueda y se está buscando a *x*, se ha terminado la búsqueda; esto es cierto suponiendo que las llaves de búsqueda son únicas. Si se busca una llave menor que *x*, entonces debe encontrarse en el subárbol izquierdo. Similarmente, si se busca una llave mayor que *x*, debe buscarse en el subárbol derecho.

Así, puede procederse recursivamente hasta encontrar la llave buscada. El siguiente algoritmo de búsqueda supone que no existen llaves duplicadas y regresa una referencia al nodo donde se encuentra la llave de búsqueda o el valor de nil si la llave no se encuentra en el árbol.

Ejemplo:

Considere el siguiente árbol binario de búsqueda:



Si se busca la llave 7 en el árbol se compara 7 con 10 y se desciende por la izquierda; se compara 7 con 8 y se desciende por la izquierda; al comparar 7 con 5 se desciende por la derecha y se encuentra la llave.

Ejemplo de un programa en C que maneja ABB

```
/*
  PROGRAMA QUE GENERA UN ÁRBOL DONDE LA INFORMACION DEL NODO ES UNA CADENA.
  LA SALIDA DEL ÁRBOL ES UN RECORRIDO INORDEN

*/

#include "malloc.h"
#include "stdlib.h"
#include "string.h"
#include "conio.h"
#include "stdio.h"

 struct nodo {

 char info[10];
 struct nodo *izq;
 struct nodo *der;
 };

//Funciones de insercion
 void ins_izq (struct nodo *p,char n[10]);
 void ins_der (struct nodo *p, char n[10]);
 void inorden (struct nodo *p);


 void main()
 {

  struct nodo *raiz,*p,*q;

  int conodos=0;
  char h,n[10];


 printf("\n\n\t [ . ] PARA TERMINAR EL INGRESO DE DATOS AL ARBOL\n");
 printf( "\n\tEl L ARBOL SERA LEIDO EN FORMA INORDEN\n");
 printf("\t_____\n");
 //lectura del nodo raiz
 printf("\n Dar una cadena\n");
 gets(n);
 h=n[0];
```

```
//Creacion del nodo raiz
raiz = (struct nodo *)malloc (sizeof (struct nodo));

 strcpy(raiz->info,n);
 raiz->izq = NULL;
 raiz->der = NULL;

//contador para el no. de nodos
 conodos+=1;

 printf("\n Dar una cadena\n");//Lectura del siguiente nodo
 gets(n);

 while (n[0] != '.')
 {
   //asignacion de apuntadores
  p = q = raiz;
  while ( q != NULL )
   {
     p = q;
              if (strcmp(n,p->info)< 0 )
        q = q->izq;

              else
        q = q->der;
   }
 conodos+=1;

// si el nodo que lee es menor que la raiz
// se realiza la insercion por la izq sino lo hace por la derecha
    if  (strcmp(n,p->info)< 0 )
        ins_izq (p,n);
  else ins_der (p,n);

   //Se pide le suiente nodo
  printf("\nDar una cadena\n");
  gets(n);
  }//fin del while que permite digitar todos los elementos

  printf("\n\n\tSalida del ARBOL (inorden): \n ");
//funcion de impresion
 inorden (raiz);
 printf("\n\n\tEl número de nodos son:   ") ;
 printf("  %d  ",conodos);

 getch();
}



 void ins_izq (struct nodo *p,char n[10])
 {
  struct nodo *nuevo;
  nuevo= (struct nodo *)malloc (sizeof (struct nodo));
  strcpy(nuevo->info,n);
  nuevo->izq = NULL;
  nuevo->der = NULL;
  p->izq = nuevo;
 }


 void ins_der (struct nodo *p,char n[10])
 {
  struct nodo *nuevo;

nuevo=(struct nodo *)malloc (sizeof (struct nodo));
```

```
 strcpy(nuevo->info,n);
 nuevo->izq = NULL;
 nuevo->der = NULL;
 p->der = nuevo;
}


void inorden (struct nodo *p)
{
 if (p!=NULL)
 {
   inorden(p->izq);
   printf("\n\t _  %s ",p->info);
   inorden(p->der);
 }
}
```

Eliminación de un nodo en un ABB

Existen 2 casos. Primero, si el nodo a eliminar es una hoja o tiene un único descendiente, lo único que hay que hacer es asignar un enlace del nodo padre el descendiente de nodo a eliminar. Segundo, si el nodo no tiene las 2 ramas no vacías (es decir tiene hijos), en este caso hay dos opciones de eliminación:

a) Reemplazar el dato del nodo por la menor de las claves mayores en su subárbol derecho.
b) Reemplazar el dato del nodo pro la mayor de las claves menores en su subárbol izquierdo. En esta opción, como las claves menores están en la rama izquierda, se baja al primer nodo de la rama izquierda y como la clave mayor está en la rama derecha, se continúa bajando por la rama derecha hasta alcanzar el nodo hoja. Este es el mayor de los menores que reemplaza a la clave del nodo a eliminar.

Ejemplos:

Se desea Eliminar el nodo 60. Aplicando la eliminación por el mayor de los menores, el nodo 55 será ahora la raíz.

Al eliminar el nodo 55 no hay ningún cambio de padre ya que el nodo es hoja.

Eliminado el nodo 5 el nuevo padre será el nodo 3

Ese desea eliminar el nodo 11 por la opción de el menor de sus mayores la nueva raíz es el nodo 12.