Arboles

Definición: Es una estructura acíclica que, en algún sentido puede ser considerado el siguiente paso en la jerarquía de complejidad estructural. Los árboles tienen son de amplia aplicación en el campo de la computación y son utilizados como estructuras de datos en distintas áreas como por ejemplo:

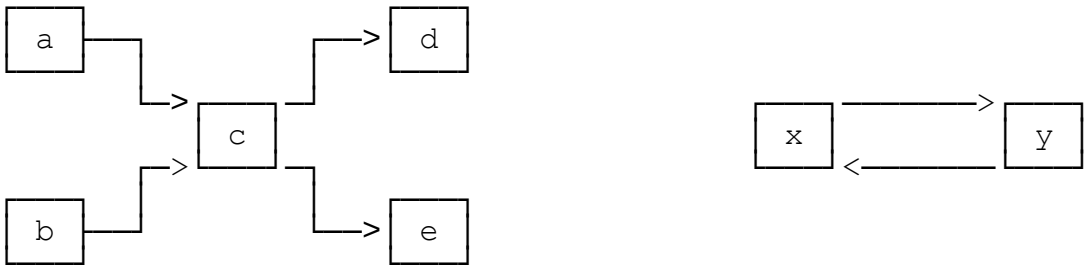
- **Arboles parse,** en la teoría de los compiladores
- **Arboles de búsqueda,** en recuperación de la información
- **Arboles de decisión,** en inteligencia artificial
- **Arboles directorios,** en sistemas operativos

Definiciones formales:

1) un árbol es un grafo acíclico finito T (P, E) tal que

|P| = |E| + 1 => todo arco es desconectante. **Ejemplo :**

T_1 **T_2**



T_3 **T_4**

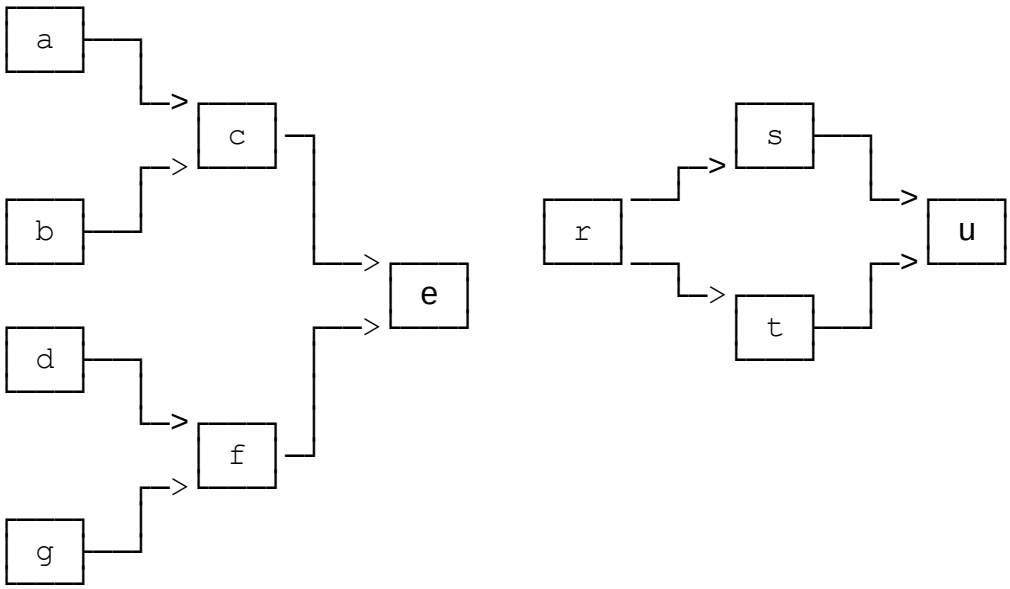


Figura 5.1.

Los árboles T_1 y T_3 son árboles mientras que T_2 y T_4 no lo son

2) Un grafo es un árbol <=> entre dos nodos cualesquiera existe un walk único.

Subárbol

Dado un grafo T que cumple con las condiciones de árbol, decimos que T' es un subárbol del árbol T si T' es un subgrafo conectado. De otra manera, si T' es en sí mismo un árbol

Arboles Principales

Recordando el concepto de ideal derecho $\overline{R(x)}$ e ideal izquierdo $\overline{\mathbf{L_{(x)}}}$ donde se constituyen los ideales algebraicos respecto de la relación de paso:

$y \in \overline{R_{(x)}}$ <=> existe $\rho_{(x,y)}$

$y \in \overline{L_{(x)}}$ <=> existe $\rho_{(y,x)}$

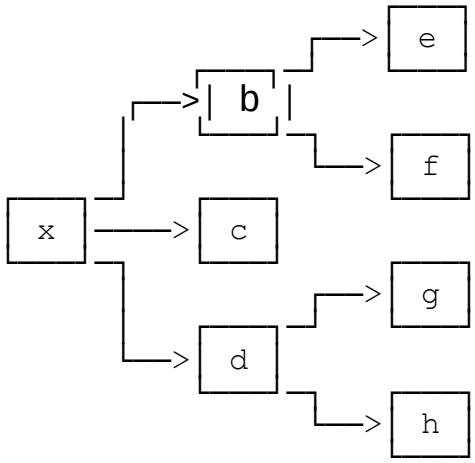
Sea T(P, E) un árbol con $x \in P$, entendemos por subárbol principal derecho en x, denotado $\mathbf{T^d_x}$, al subgrafo determinado por el ideal principal derecho en x. De la misma manera, entendemos por subárbol principal izquierdo en x, al subgrafo determinado por el ideal principal izquierdo en x. Si además, el subárbol derecho o izquierdo en x, coincide con el árbol original entonces se dice que **el árbol es principal derecho o izquierdo en x**, siendo **x el nodo principal del árbol**. Por lo tanto :

$\mathbf{T^d_x}$ es un árbol ppal. derecho en x si existe $x \in P / \mathbf{T = T^d_x}$

$\mathbf{T^i_x}$ es un árbol ppal. izquierdo en x si existe $x \in P / \mathbf{T = T^i_x}$

Si bien el ideal principal derecho o izquierdo, determina un conjunto de puntos, queda sobreentendido que el subgrafo determinado por el ideal derecho o izquierdo incluyen las relaciones, pues se pretende obtener la definición de subárbol principal a partir del concepto algebraico de ideales.

Arbol Principal Derecho -*Características*



1) x es minimal => $|L_{(x)}| = 0$

2) x se denomina **raíz** y es única

3) e, f, g, h son maximales => $|R_{(e)}|=|R_{(f)}|=|R_{(g)}|=|R_{(h)}|=0$ se los denomina **hojas del árbol**

4) $|L_{(y)}|=1$ para todo y P /y<>x

Figura 5.2.

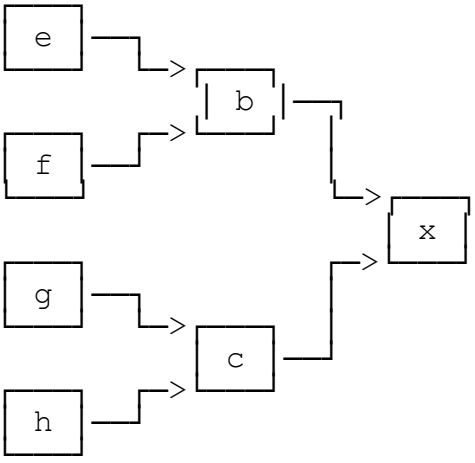
Definición : Un árbol T es principal derecho <=>

1) existe $x \in P$ / $|L_{(x)}| = 0$ siendo x único

2) se cumple que $\overline{R_{(x)}} = P$

3) se cumple que $|L_{(y)}| = 1$ para todo $y \in P$ / y <> x

Arbol Principal izquierdo -*Características*



1) x es maximal => $|R_{(x)}| = 0$

2) x es único

3) $|R_{(y)}|=1$ para todo $y \in P$ /y<>x

Figura 5.3.

Definición : Un árbol T es principal izquierdo <=>

1) existe x ∈ P / $|R_{(x)}|$ = 0 siendo x único

2) se cumple que $\overline{L_{(x)}}$ = P

3) se cumple que $|R_{(y)}|$ = 1 para todo y ∈ P / y <> x

Grado de un árbol

Esta dado por el grado de salida del nodo con mayor grado de salida. **Ejemplo:**

El grado de salida en el árbol de la **Figura 5.2.** es **3** y está dado por el grado de salida del nodo x, pues es el nodo con mayor grado de salida.

Si el árbol es de grado **2**, se lo llama **binario**.
Si el árbol es de grado **3**, se lo llama **ternario**.
Si el árbol es de grado **r**, se lo llama **r-ario**.

Profundidad de un nodo

Es la longitud de paso entre la raíz y el nodo considerado.
Ejemplo :

La profundidad del nodo **e** en el árbol de la **Figura 5.2.** es **2**
La profundidad del nodo **b** es **1**. Generalizando:

Profundidad del nodo z = $|\rho_{(x,z)}|$ siendo x la raíz del árbol

Niveles de un árbol

El nivel en un árbol es una clase de equivalencia determinada por los nodos que tienen igual profundidad. **Ejemplo:**

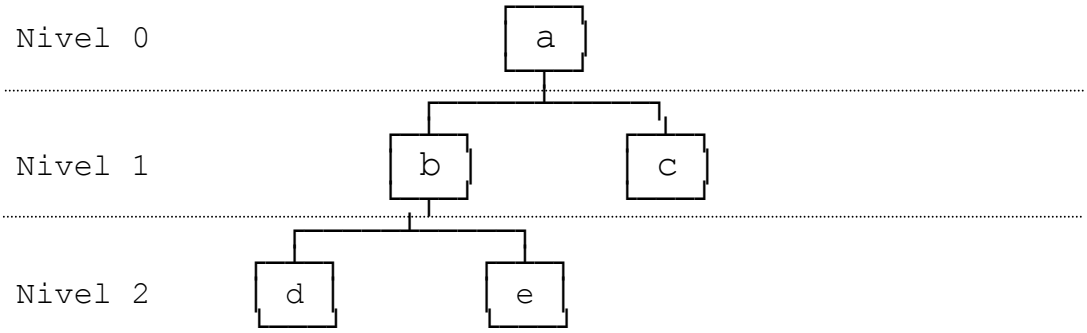


Figura 5.4.

Arbol completo

Un árbol es completo cuando todos sus nodos no maximales tienen igual grado de salida**. Ejemplo:**

El árbol de la **Figura 5.2. no es completo**.
El árbol de la **Figura 5.4. es completo**.

Arbol lleno

Un árbol es lleno cuando es un árbol completo y todos sus nodos maximales tienen igual profundidad. Generalizando:

1) Sea **r** el grado de salida del árbol

$|R(y)| = r$ para todo $y \in P$ .and. y no $\in \mathbf{Max_P}$.

2) para todo $z \in \mathbf{Max_P}$ => $|\rho_{(x,z)}|$ es la misma para todo z, siendo x la raíz del árbol.

Cardinalidad de un árbol lleno

La cantidad máxima de nodos que puede tener un árbol lleno está dado por la siguiente fórmula:

$$|P| = \sum_{i=0}^{k} r^i$$

Donde :

|P| es la **cardinalidad máxima** de un árbol lleno
k es el **último nivel** del árbol
r es el **grado de salida** del árbol

Ejemplo :

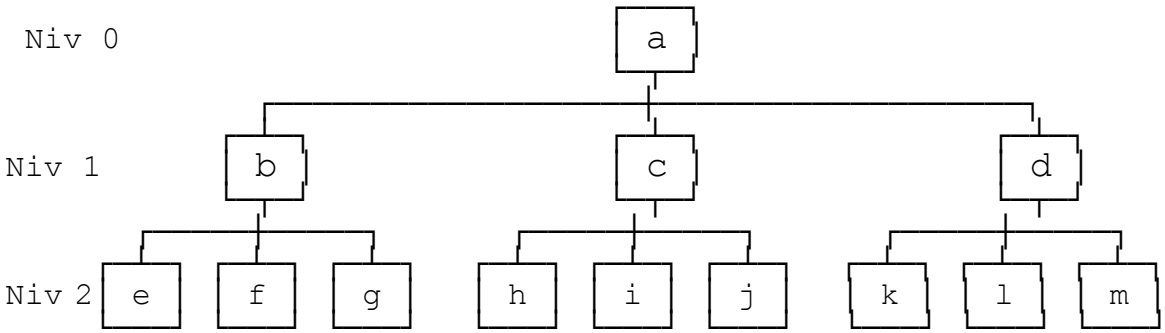


Figura 5.5.

$$\mathbf{k} = 2, \mathbf{r}=3 \Rightarrow |\mathbf{P}| = \sum_{\mathbf{i=0}}^{\mathbf{2}} \mathbf{3^i} \Rightarrow |\mathbf{P}| = (3^0 + 3^1 + 3^2) = \boxed{13}$$

Operaciones sobre árboles

Barridos: Un barrido es un orden definido sobre un conjunto de nodos. Es una operación que aplicada sobre un árbol principal derecho, devuelve una estructura lineal.

Tipos de barridos:

a) Preorden
La ejecución del barrido preorden, en forma resumida, tiene el siguiente procedimiento:

1) Informar raíz
2) Visitar subárbol izquierdo
3) Visitar subárbol derecho

1) Al comenzar el barrido, se accede a la raíz del árbol mediante un puntero externo y se **informa** el nodo raíz, es decir, se muestra el valor del identificador del nodo.
2) El paso siguiente es **visitar** el subarbol izquierdo del nodo raíz y aplicar el barrido preorden a ese subarbol. El término visitar significa acceder a la raíz de un subarbol, con la dirección del hijo izquierdo del nodo que se está analizando. El proceso continúa hasta que el campo que contiene la dirección del hijo izquierdo del nodo que se está considerando, tenga valor nil.
3) Por último, visitar el subarbol derecho y aplicar el barrido preorden a ese subarbol, es decir, informar la raíz **(paso 1)** y seguir con el análisis del subarbol izquierdo **(paso 2)**

Pseudocódigo: Para resolver el barrido preorden, es necesario utilizar una pila como estructura auxiliar, para guardar las direcciones de los nodos analizados, a medida que se visitan los subarboles izquierdos y derechos. En el algoritmo se ha utilizado un árbol binario.

Definición de las variables utilizadas.

Diseño de la pila
reg-pila
direraiz : pointer con la dirección de la raíz del subarbol visitado
reganter : pointer con la dirección del registro anterior de la pila

reg-arbol
identifi : string con el identificador del nodo
hijo_izq : pointer con la dirección del hijo izquierdo
hijo_der : pointer con la dirección del hijo derecho

variables
apu_pila : pointer al último elemento ingresado en la pila
apu_arbol : pointer a la raíz del árbol
dir_pila : pointer al registro actual de la pila
dir_arbol : pointer al registro actual del árbol

```
apu_pila  := nil;
dir_arbol := apu_arbol;
do while dir_arbol .not. = nil
   repeat
      write (dir_arbol^.identifi);
      new(dir_pila);
      dir_pila^.direraiz := dir_arbol;
      dir_pila^.reganter := apu_pila;
      apu_pila  := dir_pila;
      dir_arbol := dir_arbol^.hijo_izq;
   until dir_arbol = nil
   do while dir_arbol = nil .and. apu_pila .not. = nil
      dir_arbol := apu_pila^.direraiz
      var_pila  := apu_pila^.reganter
      dispose(apu_pila)
      apu_pila  := var_pila;
      dir_arbol := dir_arbol^.hijo_der
   enddo
enddo
```

b) Entreorden o simétrico

El barrido simétrico, tiene el siguiente procedimiento:

1) Visitar subárbol izquierdo
2) Informar raíz
3) Visitar subárbol derecho

1) Al comenzar el barrido, se accede a la raíz del árbol mediante un puntero externo y se visita el subárbol izquierdo mediante el campo hijo izquierdo. A este subarbol, se aplica el barrido simétrico nuevamente, es decir, se visita el subárbol izquierdo. Este proceso se repite hasta que el campo hijo izquierdo tenga valor nil.
2) El paso siguiente es **informar** el nodo raíz, es decir, se muestra el valor del identificador del último nodo leído.
3) El siguiente paso es visitar el subarbol derecho y aplicar el barrido simétrico a ese subarbol, es decir, visitar el subárbol izquierdo hasta que el campo hijo izquierdo sea nil **(paso 1)** e informar la raíz **(paso 2)**

c) Postorden

El barrido postorden, tiene el siguiente procedimiento:

1) Visitar subárbol izquierdo
2) Visitar subárbol derecho
3) Informar raíz

1) Al comenzar el barrido, se accede a la raíz del árbol mediante un puntero externo y se visita el subárbol izquierdo mediante el campo hijo izquierdo. A este subarbol, se aplica el barrido postorden nuevamente, es decir, se visita el subárbol izquierdo. Este proceso se repite hasta que el campo hijo izquierdo tenga valor nil.
2) El paso siguiente es visitar el subárbol derecho mediante el campo hijo derecho. A este subarbol, se aplica el barrido postorden nuevamente, es decir, aplicar nuevamente el **paso 1**. Estos dos pasos se repiten hasta que tanto el hijo derecho como el hijo izquierdo tienen valor nil.
3) El paso siguiente es **informar** el nodo raíz, es decir, se muestra el valor del identificador del último nodo leído.

Arboles de expresión:

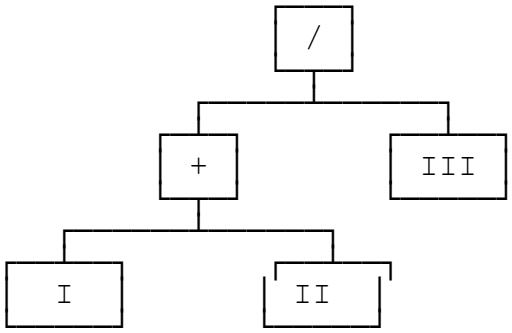
Dada la expresión:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

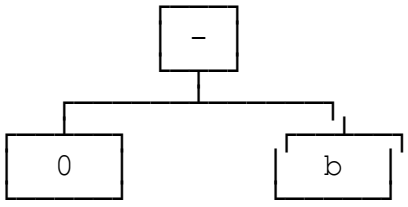
es posible representar esta expresión por medio de un árbol binario. Primero se agrupan las operaciones en función de su precedencia aritmética. Así, la expresión anterior puede tener la siguiente **notación infijo** :

```
( - b ± √ ( b ^ 2 - 4 * a * c ) ) /  ( 2 * a )
  └┘   └──────────────────────┘     └──────┘
  (I)          (II)                  (III)
```

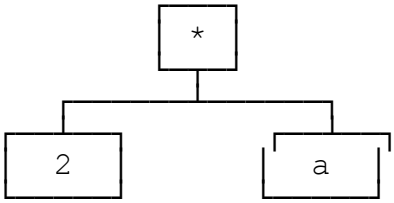
La expresión quedó compuesta por tres expresiones(1) (I, II, III) y dos operadores (±, /). Para facilitar la explicación, se reemplazará el operador ± por +. Por lo tanto, la expresión podría representarse por un árbol binario donde cada raíz de un subárbol, guardaría operadores, y cada hijo derecho e izquierdo de dicha raíz guardaría las expresiones(1). En el ejemplo:



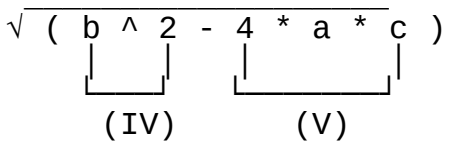
A su vez, cada una de las expresiones(1) anteriores, puede llevarse a una expresión menor que pueda ser representada a traves de un árbol. Por ejemplo,la expresión(1) (I) = -b podría representarse como (0-b), es decir:



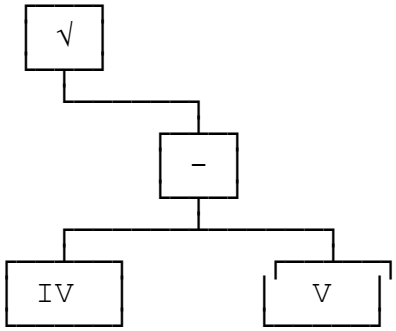
La expresión(1) (III) = (2*a)



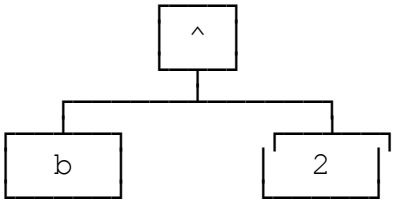
Si bien la expresión(1) (II) es más compleja, tiene igual resolución.



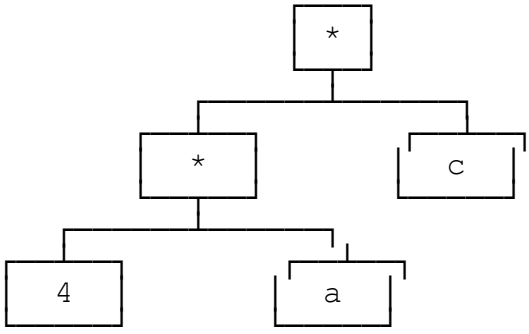
Es decir se lleva a una expresión de menor nivel que permita seguir desagregando operandos y operadores. La expresión(1) (II) podría representarse entonces con el siguiente árbol :



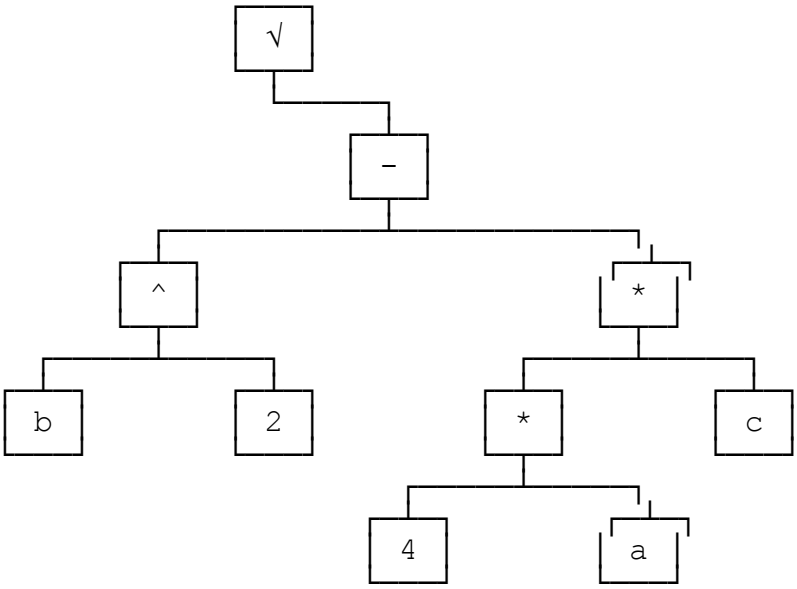
Expresión(2) (IV) = (b^2)



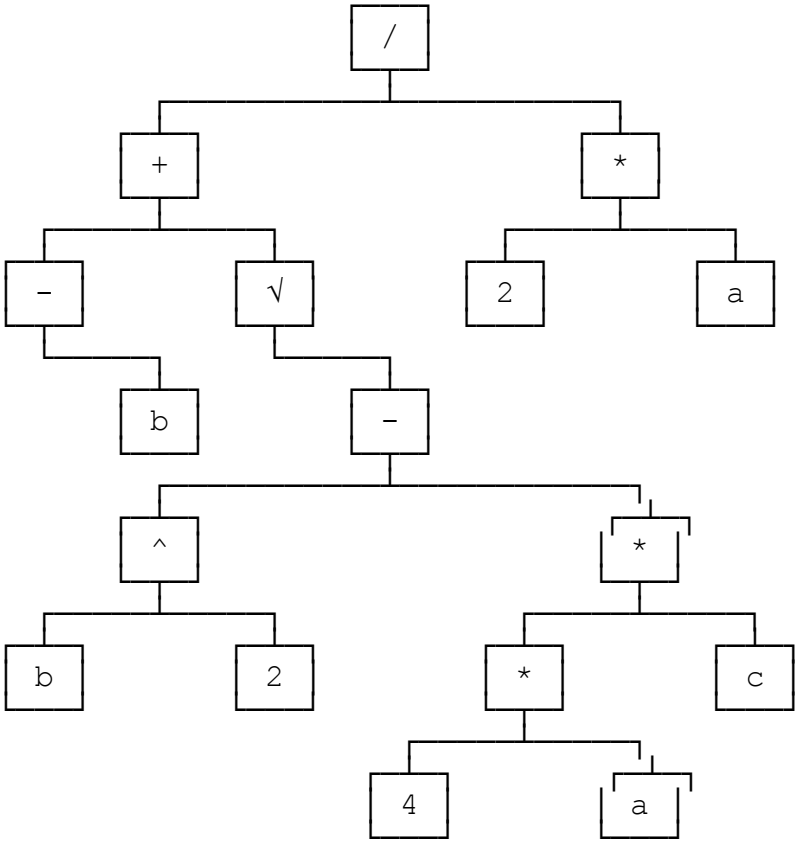
Expresión(2) (V) (4*a*c)



La expresión (1) (II) quedará entonces representada por el siguiente árbol:



En definitiva, el árbol de expresión de la ecuación original será:



Aplicando cada uno de los barridos el resultado sería:

SIMETRICO: - b + ´ b ^ 2 - 4 * a * c / 2 * a

PREORDEN: / + - b ´ - ^ b 2 * * 4 a c * 2 a

POSTORDEN: b - b 2 ^ 4 a * c * - ´ + 2 a * /

Representación de árboles r-arios:

Un árbol se llama r-ario si para todo y ∈ P se cumple que

$|R_{(y)}| <= r$

Sea el árbol

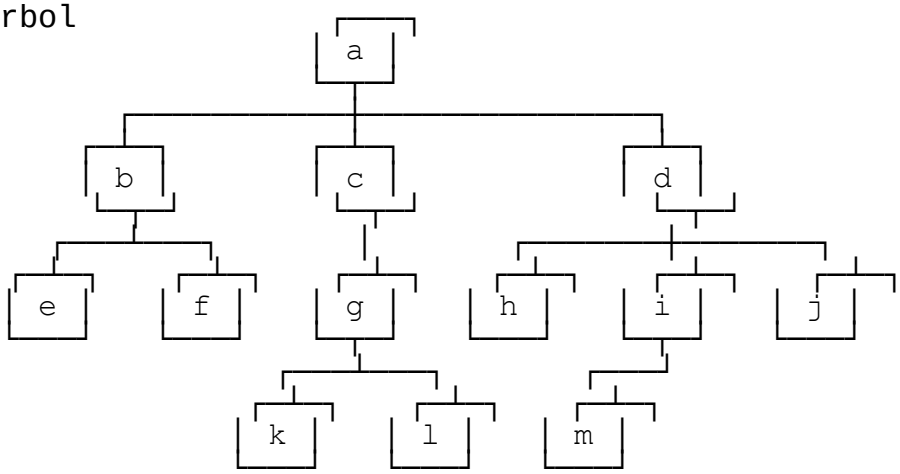


Figura 5.6.

El grado del árbol está dado por el out-degree del nodo con mayor grado de salida. En la **Figura 5.6.**, el nodo con mayor out-degree es el **nodo d**, con $|R_{(d)}| = 3$. Por lo tanto el grado del árbol es 3 y el árbol recibe el nombre de **ternario**. Mientras que el barrido simétrico no puede ser aplicado a un árbol que no sea binario, los otros barridos, aplicados al árbol del ejemplo darían el siguiente resultado:

PREORDEN: < a, b, e, f, c, g, k, l, d, h, i, m, j >

POSTORDEN: < e, f, b, k, l, g, c, h, m, i, j, d, a >

El principal problema que se presenta en los árboles r-arios, es la cantidad de espacio destinado a punteros que tiene el valor null. El espacio desperdiciado puede calcularse en forma exacta mediante la fórmula:

$$((r - 1) * |P|) + 1)$$

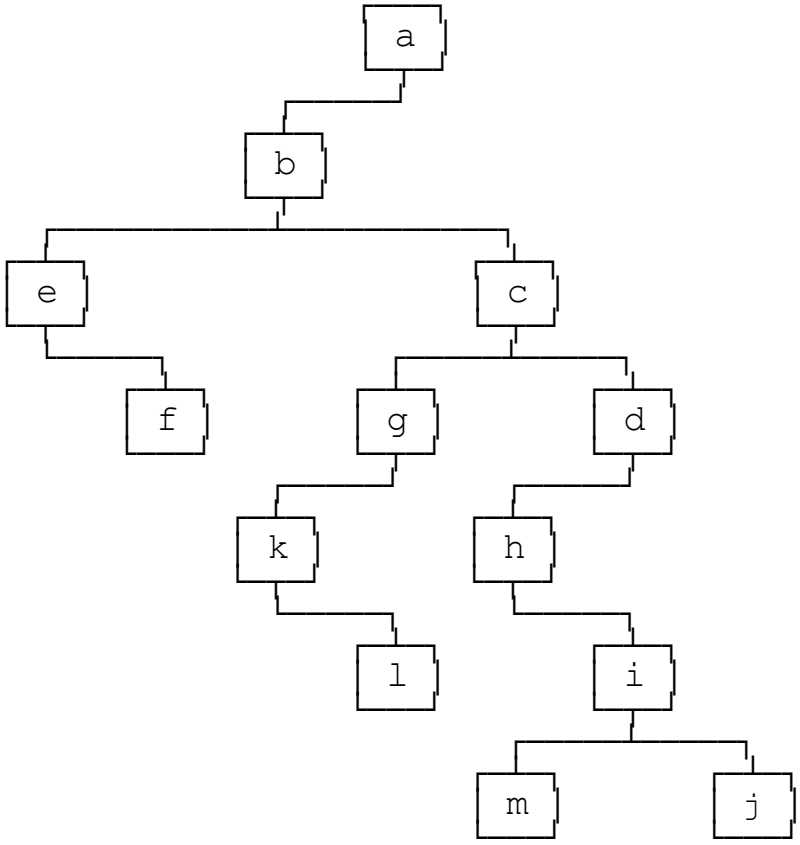
En el árbol de la **Figura 5.6.** el espacio desperdiciado es

$$((3 - 1) * 13) + 1 = \mathbf{27}$$

Por otra parte el grado de salida en una representación dinámica, puede variar instantaneamente y lo mejor sería utilizar un método de representación que; sea cual fuere el grado de salida; utilice celdas de tamaño fijo con un número de campos link predeterminados.

Transformada de Knuth

Knuth transforma un árbol r-ario en un árbol binario de forma tal que, dado un nodo, coloca como hijo izquierdo el primer elemento de su right y como hijo derecho el siguiente nodo del mismo nivel del mismo padre. En el ejemplo de la **Figura 5.6.**



Al ser este último árbol binario, le pueden ser aplicados los barridos preorden postorden y simétrico.

PREORDEN: < a, b, e, f, c, g, k, l, d, h, i, m, j >

SIMETRICO: < e, f, b, k, l, g, c, h, m, i, j, d, a >

POSTORDEN: < f, e, l, k, g, m, j, i, h, d, c, b, a >

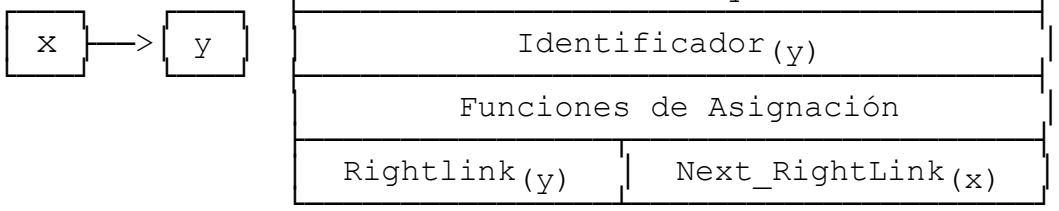
Comparando los barridos realizados al árbol original y el árbol binario resultante de haber aplicado la transformada de Knuth se concluye que:

PREORDEN original = PREORDEN Knuth

POSTORDEN original = SIMETRICO Knuth

Formato de registro para la representación de Knuth

Sea el grafo



LeftLink (v) Puntero al padre de v en el árbol original.

Identificador(v) Identificador del nodo v

Funciones de asignación: Funciones de asignación al nodo v

Rightlink (v): Puntero a su primer hijo en el árbol n-ario original.

Next RightLink(x): Puntero al próximo hermano de v, o sea al proximo nodo el cuál posea el mismo padre “x”.

Ejemplo de Programa Conversión de expresión

```
Program ConvierteExpresion;
const t = 20;
type  Puntero = ^regPila;
      regPila = record
                   Dato : char;
                   priori : integer;
                   Ant  : Puntero;
                end;
     tira     = array [1..t] of char;
var  Apunto, ApuntAnt, dir : Puntero;
     i, h, z  : integer;  entra : char; entrada, sale : tira;

Procedure Infijo ( var h : integer );
var   nivel   : integer;
      priorid : integer;
begin
   writeln('  Ingreso de la Expresion Infijo -( Por "0" finaliza)
');
   Apunto := nil;
   h      := 0;
   z      := 0;
   write('  Ingrese caracter : '); readln(entra);
   Repeat
      h := h + 1;
      entrada[h] := entra;
      write('Ingrese caracter : '); readln(entra);
   until entra = '0';
   For i := 1 to h do
   begin
      if entrada[i] = '('
         then nivel := nivel + 1;
      if entrada[i] = ')'
         then nivel := nivel - 1;
      if (entrada[i] = '+') OR (entrada[i] = '-')
         then priorid := (nivel * 10) + 1;
      if (entrada[i] = '*') OR (entrada[i] = '/')
         then priorid := (nivel * 10) + 2;
      if (entrada[i] = '*') OR (entrada[i] = '/') OR
         (entrada[i] = '+') OR (entrada[i] = '-')
         then begin
                 while ( Apunto^.priori > priorid) AND (Apunto <>
nil) do
                 begin
                    z := z + 1;
                    sale[z] := apunto^.dato;
                    ApuntAnt := Apunto;
                    Apunto := Apunto^.Ant;
                    Dispose (ApuntAnt);
                 end;
                 new (dir);
                 dir^.Dato   := entrada[i];
                 dir^.priori := priorid;
                 dir^.ant    := apunto;
                 apunto     := dir;
```

```
            end
        else
```

```
               begin
                  if (entrada[i] <> '(') AND (entrada[i] <> ')')
                     then begin
                             z := z + 1;
                             sale[z] := entrada[i];
                          end;
               end;
   end;
   while (Apunto <> nil) do
   begin
      z := z+1;
      Sale[z]  := Apunto^.Dato;
      ApuntAnt := Apunto;
      Apunto   := Apunto^.ant;
      dispose (ApuntAnt);
   end;
   writeln(' Expresion en PostFijo ');
   for i := 1 to z do
       writeln('     ',sale[i]);
end;

Begin (* Program body *)
   Infijo (h);
end.
```