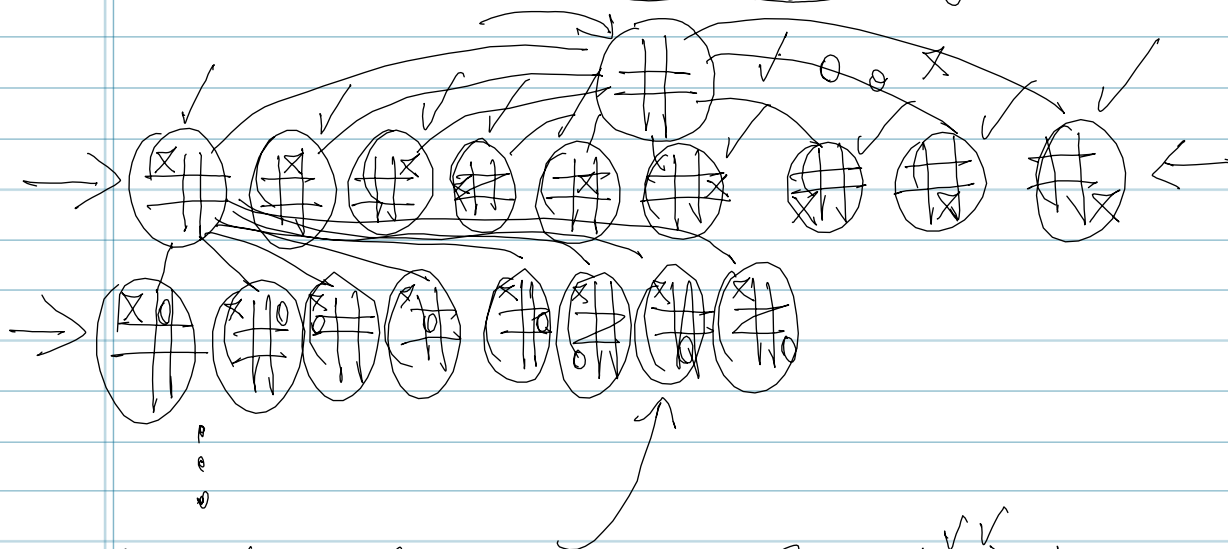


Teoría de árboles

En la práctica:

- Árbol genealógico
- Esquema de navegación
- Posibles estrategias a seguir

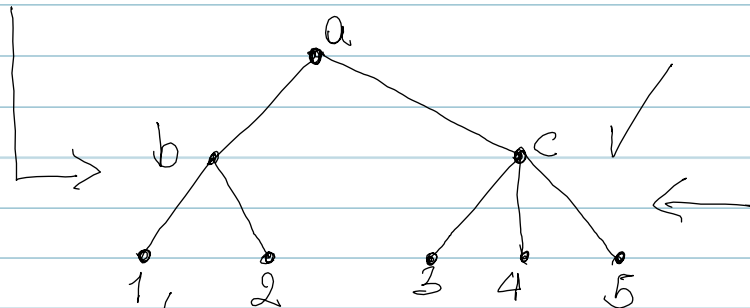


Definición Un árbol (T, r) es un grafo $T = (V, E)$, donde para cualquier par de vértices distintos de V existe una única ruta que los une. Al nodo r se le llama raíz de T y se dice que se encuentra en el nivel 1 del árbol. Si $r_1 \rightarrow r_2 \rightarrow \dots \rightarrow r_n$ es una trayectoria sobre T , se define que el vértice (r_{i-1}) es padre de (r_i) , o bien, (r_i) es hijo de (r_{i-1}) . Los hijos de la raíz se encuentran en el nivel 2 de T , los hijos de los hijos de la raíz de T , si es que existen, se dice que están en el nivel 3 y así sucesivamente. Se define que (r_i) es un antecesor de (r_j) , si $i < j$. Dos nodos que tienen el mismo padre se denominan hermanos. Los vértices en T que no son padres, se llaman nodos terminales, finales u hijos del árbol. Un vértice no terminal es un nodo interno de T . Un K -árbol es aquel, donde cualquier padre tiene a lo sumo K hijos. Si en particular todo padre posee exactamente K hijos, el árbol se llama completo. Si $K=2$ un K -árbol se denomina árbol binario. La altura se define como el nivel máximo alcanzado, menos uno.

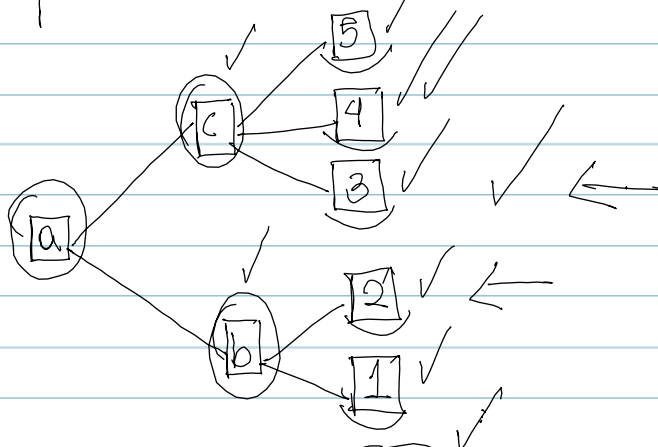
In Mathematica:

Tree Graph
Tree Plot

Tree Graph [$a \rightarrow b, a \rightarrow c, b \rightarrow 1, b \rightarrow 2, c \rightarrow 3, c \rightarrow 4, c \rightarrow 5$], VertexLabels \rightarrow "Name", ImagePadding \rightarrow 10]



Tree Plot [$a \rightarrow b, a \rightarrow c, b \rightarrow 1, b \rightarrow 2, c \rightarrow 3, c \rightarrow 4, c \rightarrow 5$], Left, VertexLabeling \rightarrow True, ImagePadding \rightarrow 10]

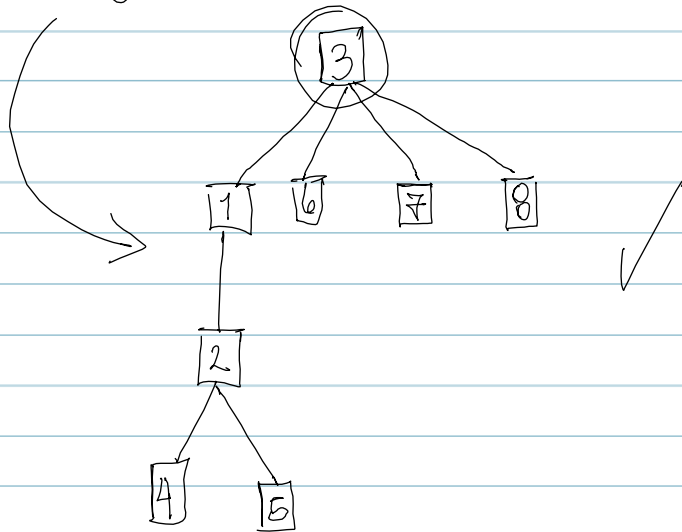


Tree Graph \rightarrow root, For example:

Tree Graph [$1, 2$, Property [$\{3, \text{"root"} \rightarrow \text{True}\}$, 4, 5, 6, 7, 8], $1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 4, 2 \rightarrow 5, 3 \rightarrow 6, 3 \rightarrow 7, 3 \rightarrow 8$], VertexLabels \rightarrow "Name", ImagePadding \rightarrow 10]

root = First@Select [VertexList [$\{1\}$], PropertyValue [$\{1\}$, #, "root"]]

Tree Plot [T], Automatic, root, Vertex Labeling → True,
Image Padding → 10]



Tree Plot accepts: Directed Edges → True

} Graph [G]
 } Show Graph [G]

Random Tree [n] → Combinatorica

For example:

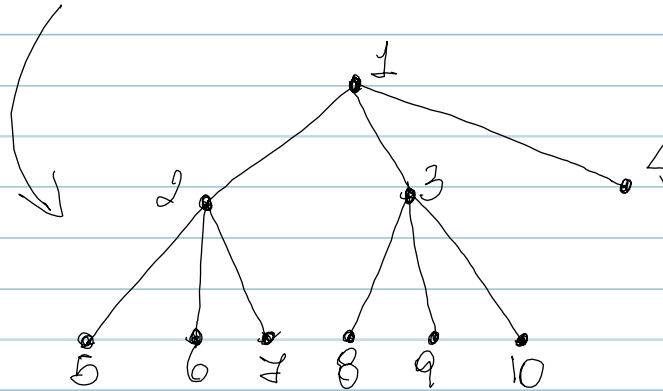
<< Combinatorica!
 Show Graph [Random Tree [10], Vertex Number → True, Plot Range
 → 0.1]
 Quit[]

un k-árbol con n vértices:

Kary Tree [k, n]

for ejemplo:

→ Kary Tree [10, 3], Vertex Labels → "Name", Image Padding → 10]

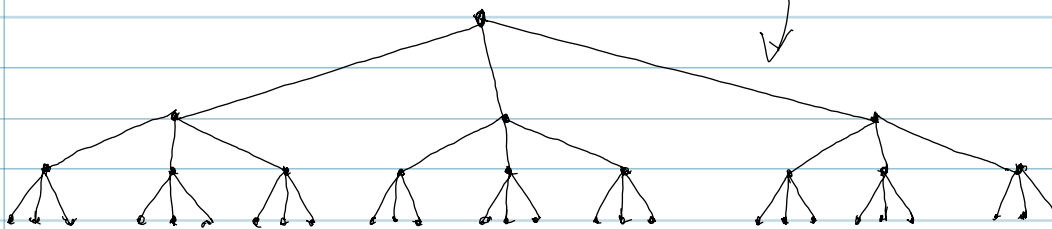


Los k-árboles completos:

Complete Kary Tree [n, k]

Por ejemplo:

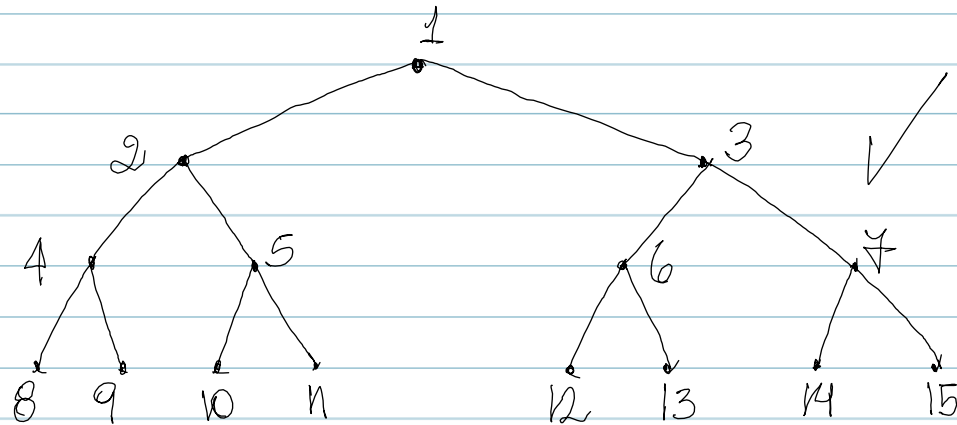
→ Complete Kary Tree [4, 3]



Otros árboles fundamentales son los árboles binarios. En Matemática:

→ Kary Tree [n]
Complete Kary Tree [n]

Complete Kary Tree [4], VertexLabels \rightarrow "Name", ImagePadding \rightarrow 10]



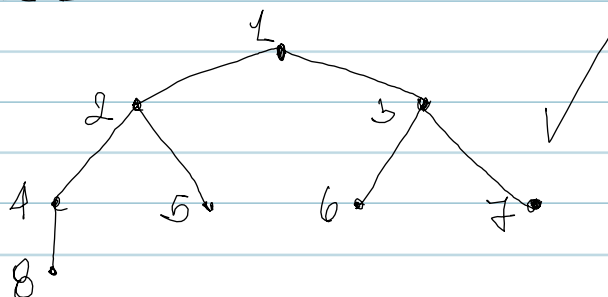
Observación: Complete Kary Tree ✓ $n-1$
 n niveles
 Combinatoria
 Kary Tree [n, k]

El paquete Combinatorica:

↓
Complete Binary Tree [n]
Kary Tree [n]

For example:

[[Combinatorica]
 ShowGraph [Complete Binary Tree [8], VertexNumber \rightarrow True,
 PlotRange \rightarrow 0.1]
 Quit[]

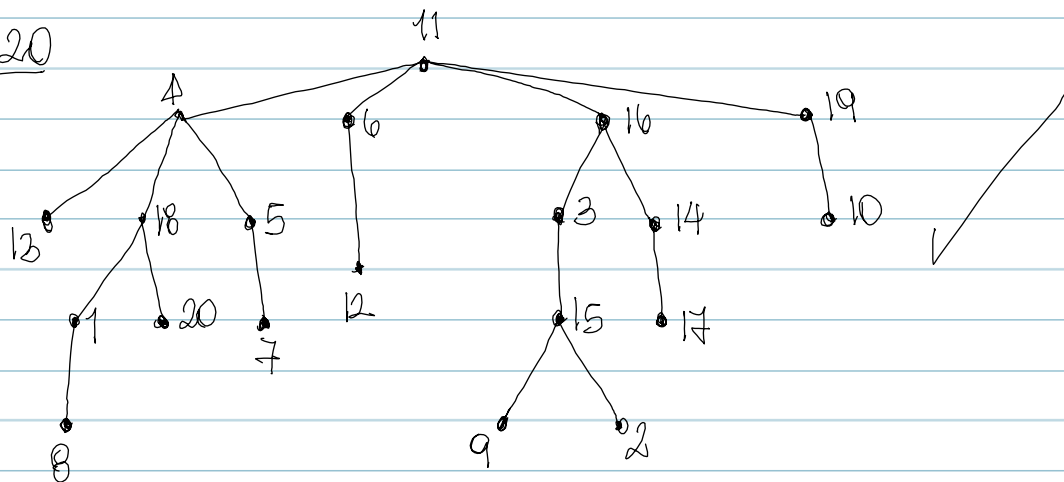


Ejemplo A través de la instrucción Random Graph, desarrolle una función que forme árboles pseudoaleatorios con (n) vértices.

Ya se había creado la función ArbolSAConexo

```
ArbolSA[n] := Module[{g}, g = Random Graph[{n}, n-1],
  VertexLabels -> "Name", ImagePadding -> 10];
→ White[Connected GraphQ[g]] == False,
→ g = Random Graph[{n}, n-1], VertexLabels -> "Name",
  ImagePadding -> 10]; Return[g]
```

n = 20



Otra sentencia interesante es: Tree GraphQ[g]

Ejemplo Verifique la correctitud de la función ArbolSA utilizando el comando Tree GraphQ.

En Mathematica:

```
→ h = 0;
→ For[i = 1, i < 1000, If[Tree GraphQ[ArbolSA[i]]] == False, h = 1,
  Break[]]; i++];
If[h == 0, Print["En todos los casos se formó un árbol"],
Print["En al menos un caso no se formó un árbol"]]
```

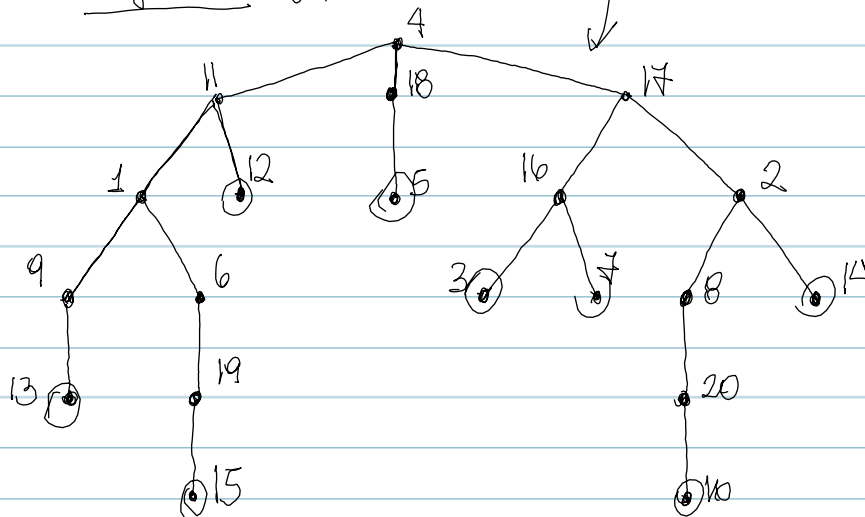
Ejemplo Diseñe una función en Mathematica que devuelva la lista de todos los nodos terminales de un árbol recibido como parámetro.

En el software:

```

Hojas[T_Graph] := If[TreeGraphQ[T] == True, Module[{
  Lhojas = {}, Lv = VertexList[T]; Lg = VertexDegree[T];
  For[i = 1, i <= VertexCount[T], If[Lg[[i]] == 1, Lhojas =
  Append[Lhojas, Lv[[i]]]; i++]; Return[Lhojas]
}, Print["El parámetro ingresado no es un árbol"]]]
  
```

Al correr Hojas[T] en el árbol:



Retorna: {3, 5, 7, 10, 12, 13, 14, 15}

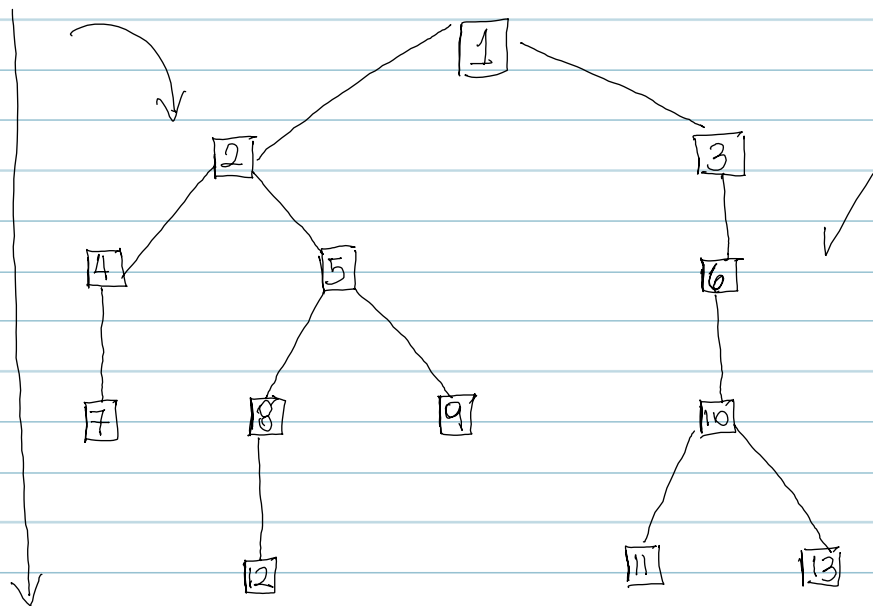
Ejemplo Elabore una función que determine la altura de un árbol.

```

Altura[T_Graph] := Module[{Lr = 2, NR = {}, R = Edges[T];
  C = R; While[C != {}, C = Composiciones[R, e];
  If[C != {}, NR = Append[NR, Lr]; Lr++];
  Print["La altura del árbol es: ", Max[NR]]]
  
```

Por ejemplo:

$n = \{1, 2\}, \{1, 3\}, \{2, 4\}, \{2, 5\}, \{3, 6\}, \{4, 7\}, \{5, 8\}, \{5, 9\}, \{6, 10\}, \{8, 12\},$
 $\{10, 11\}, \{10, 13\};$
`showGraph[`
`T = SetGraphOptions[FromUnorderedPairs[n], VertexColor -> Gray,`
`EdgeColor -> Black], VertexLabel -> True, PlotRange -> 0.1];`
 \rightarrow `TreePlot[T, Automatic, ①, VertexLabeling -> True, ImagePadding -> 10]`
Altura[T]



La altura del árbol es: 4

Ejemplo Halle con ayuda de Mathematica, el número de vértices y
lados en un k-árbol completo de n niveles

En este sentido: Vertex Count y Edge Count

$n_v = \text{Vertex Count}[\text{CompleteKary Tree}[n, k]];$

$n_e = \text{Edge Count}[\text{CompleteKary Tree}[n, k]];$

\rightarrow `Print["La cantidad de nodos en un k-árbol completo de n niveles es:`
`n_v, "` y el número de aristas corresponde a: `" n_e]`

$$\frac{-1 + k^n}{-1 + k} \rightarrow 2^n - 1 \quad \frac{-k + k^n}{-1 + k} \rightarrow 2^n - 2$$

Teorema Sea T un árbol binario completo con i vértices internos entonces
 T contiene $i+1$ hojas y $2i+1$ nodos en total.

Ejemplo Verifique el teorema anterior sobre distintos árboles binarios completos.

Al recurrir a Mathematica:

```
bandera = 0;
→ For [j = 2, j ≤ 20, i = 0; T = CompleteKaryTree[j];
  Lg = VertexDegree[T]; → For [k = 1, k ≤ VertexCount[T];
  → If [Lg[[k]] ≠ 1, i++, k++]; CantNT = 2^i - 1 - i;
  → If [CantNT ≠ i+1 || 2^i - 1 ≠ 2i+1, bandera = 1; Break[]]; j++]
→ If [bandera == 0, Print["Verificación exitosa"],
  Print["Verificación no exitosa"]]
```

Out[]

Verificación exitosa

Códigos de Huffman

MIT

David A. Huffman

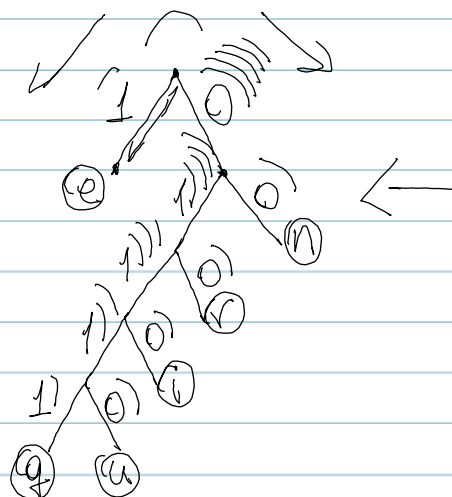
→ <https://www.computer.org/portal/web/awards/huffman>

Tal es el caso: ASCII

enrique ⇒ $\begin{cases} e:2 & i:1 \\ n:1 & q:1 \\ r:1 & u:1 \end{cases}$

$\begin{bmatrix} 1 & 00 & 010 & 0110 & 0111 & 0110 \end{bmatrix}$
 e n r i q u e

1000100100100111011101



Teorema (Algoritmo de códigos de Huffman optimizado)

Si $L = \{f_1, f_2, \dots, f_n\}$ es una lista de frecuencias de un alfabeto de longitud n , entonces:

① Sean \hat{f}_i y \hat{f}_j las dos frecuencias menores de L , tómese $n_{ij} = \hat{f}_i + \hat{f}_j$ y sustituya en L , \hat{f}_i y \hat{f}_j por n_{ij} . Se forma un árbol binario con raíz n_{ij} e hijos a y b , correspondientes a las frecuencias \hat{f}_i y \hat{f}_j , respectivamente.

② Si la longitud de L es 1 se ha terminado. El árbol buscado se construye uniendo los árboles obtenidos en el proceso, trabajando hacia atrás.

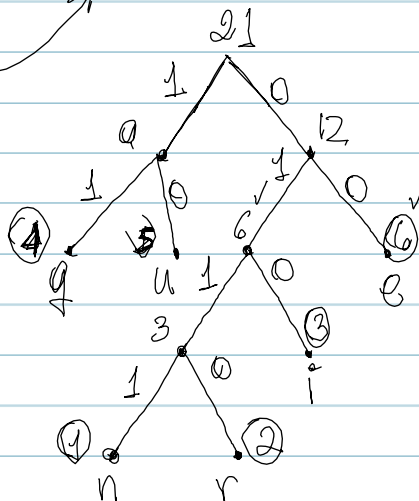
③ Vuelva al paso 1.

Por ejemplo: "enrique"

$n: 1$	$q: 4$
$r: 2$	$u: 5$
$i: 3$	$s: 6$

$1, 2, 3, 4, 5, 6 \rightarrow 1+2, 3, 4, 5, 6$
 $3, 3, 4, 5, 6 \rightarrow 3+3, 4, 5, 6$
 $4, 5, 6, 6 \rightarrow 4+5, 6, 6$
 $6, 6, 9 \rightarrow 6+6, 9$
 $9, 12 \rightarrow 9+12$

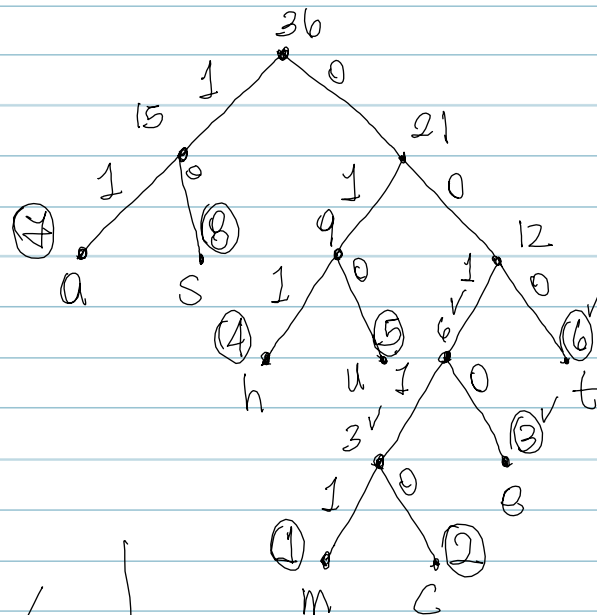
②



Supongamos ahora "massachusetts"

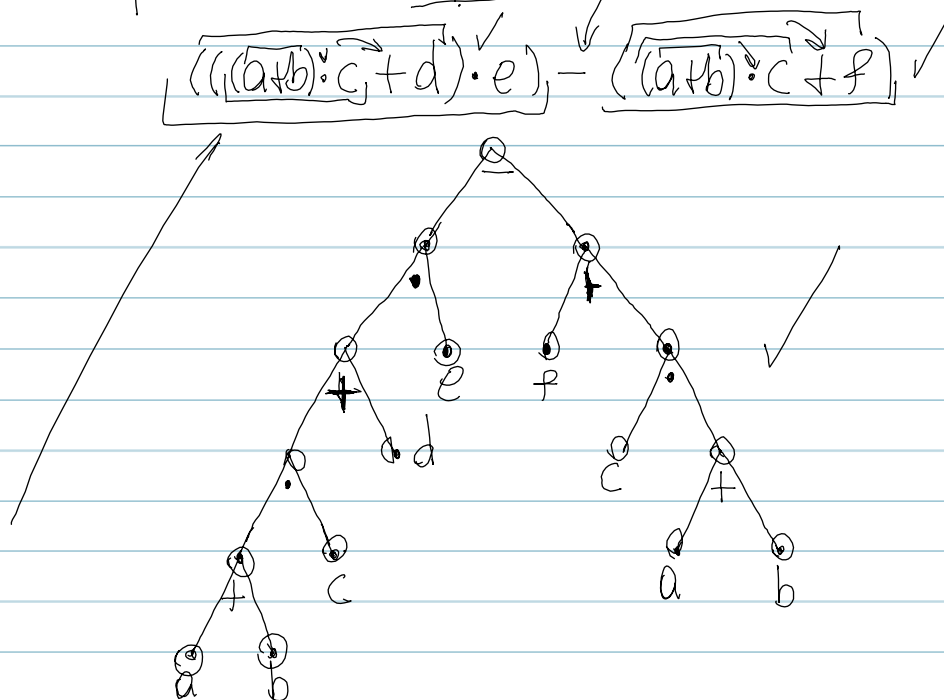
m: 1 u: 5
 c: 2 t: 6
 s: 3 a: 7 ✓
 h: 4 s: 8 ✓

1, 2, 3, 4, 5, 6, 7, 8 → 1+2, 3, 4, 5, 6, 7, 8
 3, 3, 4, 5, 6, 7, 8 → 3+3, 4, 5, 6, 7, 8
 4, 5, 6, 6, 7, 8 → 4+5, 6, 6, 7, 8
 6, 6, 7, 8, 9 → 6+6, 7, 8, 9
 7, 8, 9, 12 → 7+8, 9, 12
 9, 12, 15 → 9+12, 15
 15, 21 → 15+21
36



Arbol Huffman [Lista - List] := Module [L = Lista, Arbol = List,
 For [j = 1, j ≤ Length [Lista] - 1, L = sort [L]; Nn = L [1, 1] + L [2, 1];
 Arbol = Append [Arbol, Nn → L [1, 2]]; Arbol = Append [Arbol, Nn → L [2, 2]]
 L = Delete [L, 1]; L = Delete [L, 1]; L = Append [L, Nn, Nn]; j++
 If [j ≤ Length [Lista], Print [TreePlot [Arbol, Automatic, Nn,
 VertexLabeling → True, ImagePadding → 10]]]

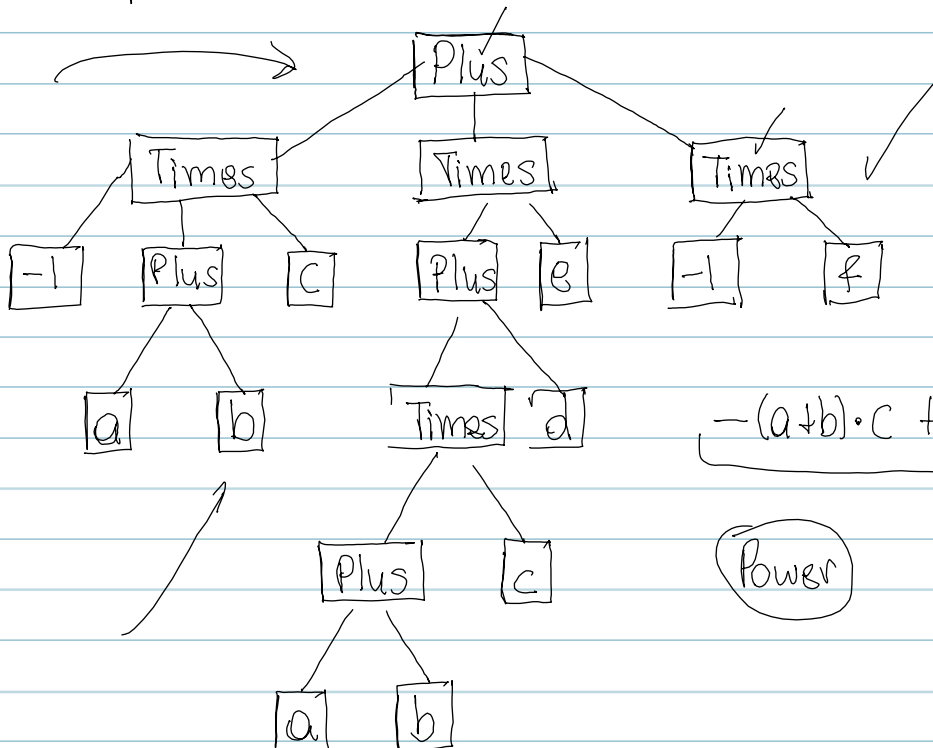
Otra aplicación de los κ -árboles



En Mathematica:

TreeForm

Por ejemplo: TreeForm[$((a+b) \cdot c + d) \cdot e - ((a+b) \cdot c + f)$]



$-(a+b) \cdot c + ((a+b) \cdot c + d) \cdot e - f$

Power

Teorema Sea $T = (V, E)$ un grafo con (n) vértices. T es un árbol si y solo si se satisface alguna de las siguientes propiedades:

- ① T es conexo y no contiene circuitos.
- ② T es conexo y posee $n-1$ aristas.
- ③ T no contiene circuitos y tiene $n-1$ lados.

Ejemplo Elabore tres funciones booleanas distintas en Mathematica, que reciban como parámetro un grafo T y determinen si T es un árbol. Tome como base las propiedades del teorema anterior.

En Mathematica:

→ Boolean1Arbol [T , Graph] := IF [ConnectedGraphQ [T] == True && AcyclicGraphQ [T] == True, Return [True], Return [False]]

Boolean2Arbol [T , Graph] := IF [ConnectedGraphQ [T] == True && EdgeCount [T] == VertexCount [T] - 1, Return [True], Return [False]]

Boolean3Arbol [T , Graph] := IF [AcyclicGraphQ [T] == True && EdgeCount [T] == VertexCount [T] - 1, Return [True], Return [False]]

Ejemplo Construya una función booleana con Mathematica que concluya si un grafo recibido como parámetro es un árbol binario.

BooleanBinario [T , Graph, raiz] := Module [$\{V\}$ = True, BooleanArbol],
BooleanArbol [T] := IF [ConnectedGraphQ [T] == True && AcyclicGraphQ [T] == True, Return [True], Return [False]]; Lg = VertexDegree [T];
→ IF [BooleanArbol [T] == True, For [i = 1, i ≤ Length [Lg],
pos = Position [VertexList [T], (raiz)]]; pos ≤ pos [i]];
→ IF [i == pos [i]], IF [Length [Lg [i]] > 3, V = False; Break []];
IF [Length [Lg [i]] > 3, V = False; Break []]; i ++]; Return [V];
Print ["El grafo ingresado no es un árbol"]]]

Ejemplo usando la instrucción Random Graph, genere una función que construya árboles binarios pseudoaleatorios con n vértices.

ArbolSA

ArbolSAB [N_] := Module [d Boolean Binario,

Boolean Binario [G-Graph, raíz_] := Module [d V1= True, Lg= VertexDegree [G], For [i=1, i≤Length [Lg], lpos=Position [VertexList [G], raíz], pos= lpos [[1]], If [i≤pos [[1]], If [Lg [[i]] ≥ 3, V1= False; Break [], If [Lg [[i]] > 3, V1= False; Break []], i++], Return [V1];

G= RandomGraph [d, n-1, VertexLabels → "Name", ImagePadding → 10];

While [Boolean Binario [G, V1] == False || ConnectedGraphQ [G] ==

False, G= RandomGraph [d, n-1, VertexLabels → "Name",

ImagePadding → 10]; Ln= VertexList [G]; Ln= ReplacePart [Ln, Property [1, "root" → True], 1]; Ll= EdgeList [G];

G= TreeGraph [Ln, Ll, VertexLabels → "Name", ImagePadding → 10];

root= First@Select [VertexList [G], PropertyValue [d, #, "root"] &];

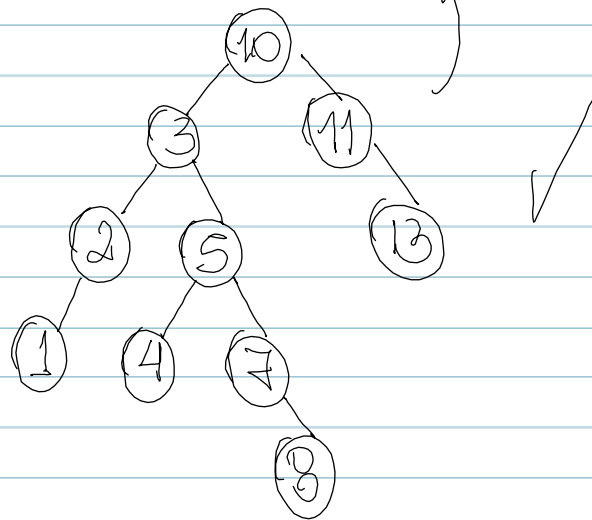
TreePlot [G, Automatic, root, VertexLabeling → True, ImagePadding → 10]

Árboles binarios de búsqueda

- Almacenar información
- Ordenar datos

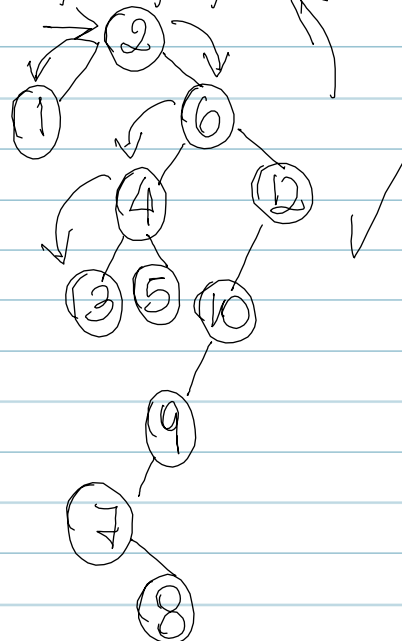
Ejemplo 1

Datos = {10, 3, 2, 5, 1, 11, 4, 7, 8, 13}



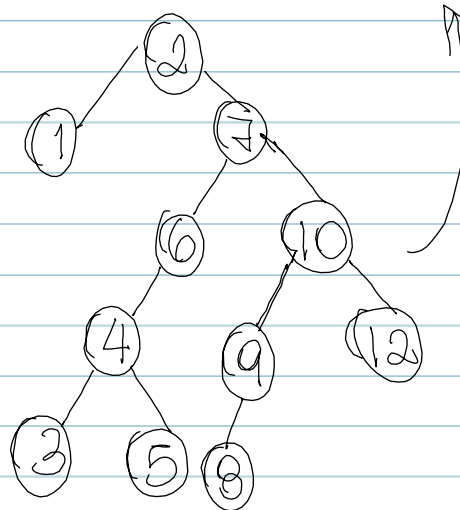
Ejemplo 2

Datos = {2, 6, 4, 3, 1, 12, 10, 9, 7, 5, 8}



Ejemplo 3

Datos = {2, 7, 6, 4, 10, 3, 1, 12, 9, 5, 8}



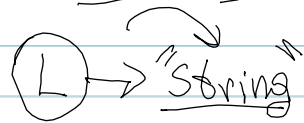
Una implementación en Mathematica:

```

W = {};
Ordena [Datos_List] :=
Module [ { CNX = {}, CND = {}, L = DeleteDuplicates [Datos] },
  IF [Length [L] > 2, For [i = 2, 1 < Length [L],
    IF [L[[i]] > L[[i-1]], CNX = Append [CNX, L[[i]]],
    CND = Append [CND, L[[i]]]; i++];
  IF [CNX != {}, W = Append [W, L[[1]] -> CNX[[1]]];
  IF [CND != {}, W = Append [W, L[[1]] -> CND[[1]]];
  Ordena [CNX]; Ordena [CND]]];

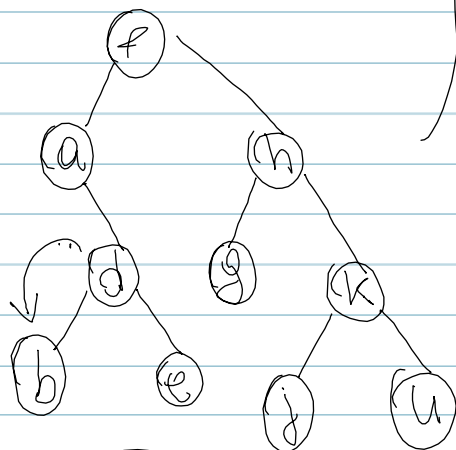
Datos = {10, 3, 2, 5, 1, 11, 4, 7, 8, 13};
Ordena [Datos]
TreePlot [W, Automatic, Datos[[1]], VertexLabeling -> True,
ImagePadding -> 10]
  
```

Otros tipos de árboles binarios de búsqueda



Por ejemplo

Datos = {f, a, d, e, h, k, g, b, j, u}



En Matemática: Ordena [Datos - List] ✓

$L[R[1]] > L[R[i]]$

$\text{Order}[L[R[1]], L[R[i]]] \rightarrow -1$

$L[R[1]] > L[R[i]]$

$\rightarrow \exists f [L[R[1]] > L[R[i]], \dots]$

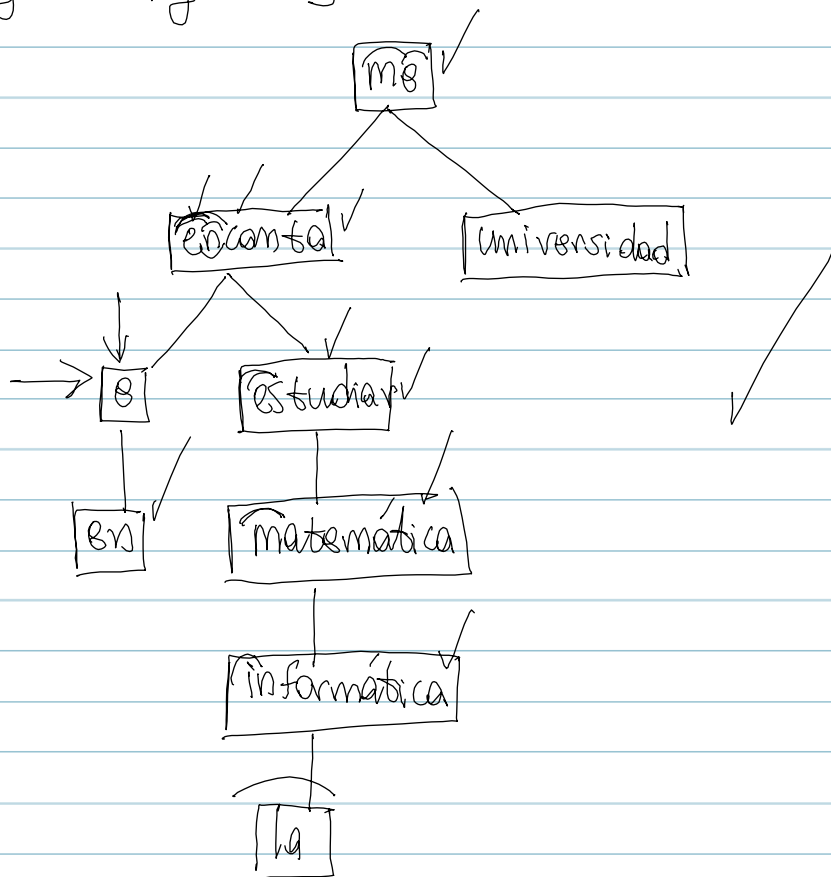
$\rightarrow \exists f [\text{Order}[L[R[1]], L[R[i]]] = -1, \dots] \checkmark$

También, se podría requerir para una frase.

Datos = String Split ["me encanta estudiar matemática e
informática en la universidad"];

Ordena [Datos]

Tree Plot [w, Automatic, Datos [[1]], Vertex Labeling → True,
Image Padding → 10]



Recorridos en un K-árbol

orden inicial ✓
orden final ✓

Teorema (Recorridos en un K-árbol) Sea $T = (V, E)$ un K-árbol. Supóngase que las ramas derivadas de la raíz de T , se enumeran en orden como $1, 2, \dots, K$, si es que existen, entonces:

1. El recorrido de orden inicial se describe mediante el siguiente procedimiento:

- Si V es vacío, regrese.
- Imprima la raíz (v) del árbol T .
- Procese la rama 1 de T llamando a este algoritmo. Procese la rama 2 de T , si es que existe, invocando a este algoritmo. Continúe hasta llegar a la última rama y regrese.

2. El recorrido de orden final se efectúa así:

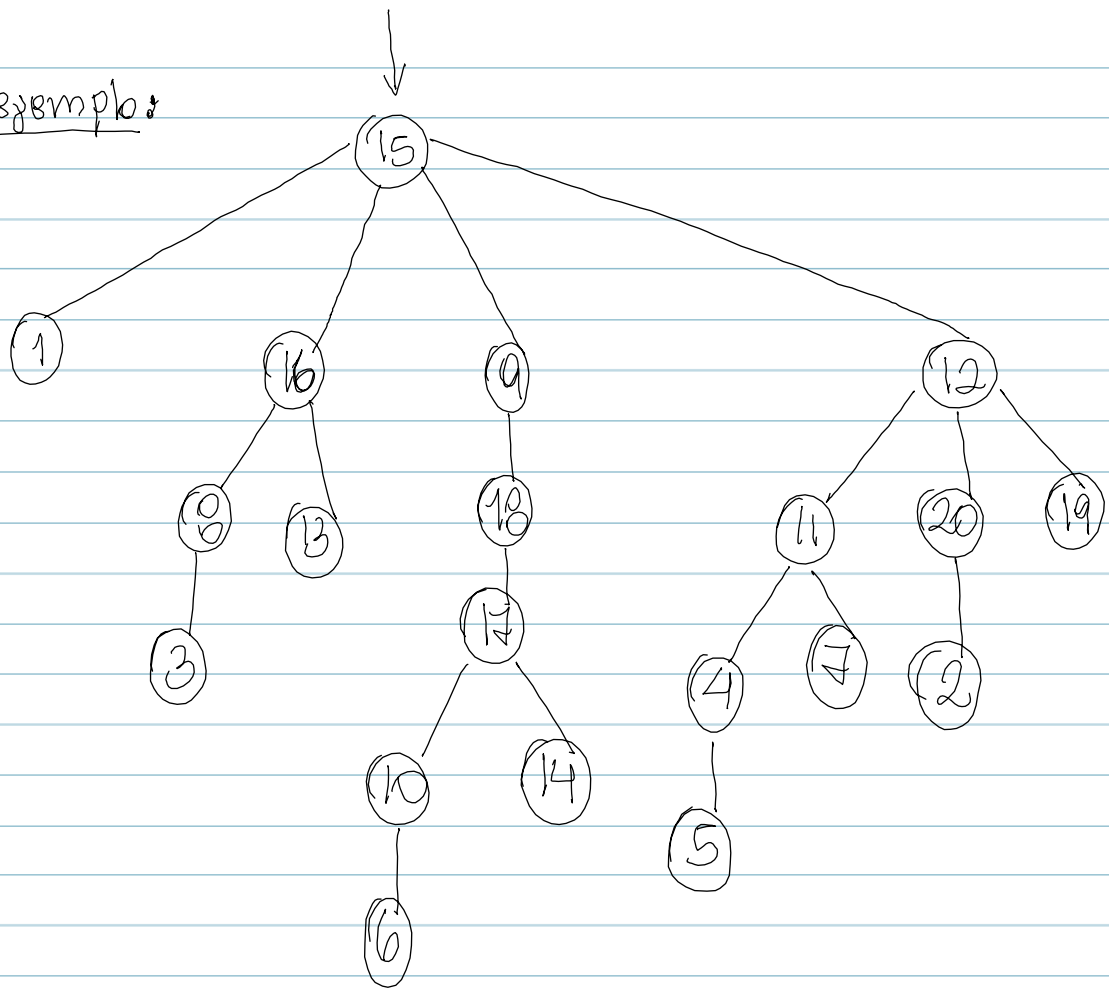
- Si V es vacío, regrese.
- Procese la rama 1 de T llamando a este algoritmo. Procese la rama 2 de T , si es que existe, invocando a este algoritmo. Continúe hasta llegar a la última rama.
- Imprima la raíz (v) y regrese.

Son métodos recursivos

En Matemática:

Depth First Search → Previsit Vertex ✓
Postvisit Vertex ✓

Por ejemplo:



usando el algoritmo prefijo:

root = 15; ✓

L = {};

DepthFirstScan [T, root, ↓ "Previsit Vertex" → (L = Append [L, #])
 Q) ↓];

Print [L] → {15, 1, 16, 8, 3, 13, 9, 18, 17, 10, 6, 14, 12, 11, 4, 5, 7, 20, 2, 19}

usando el algoritmo postfijo:

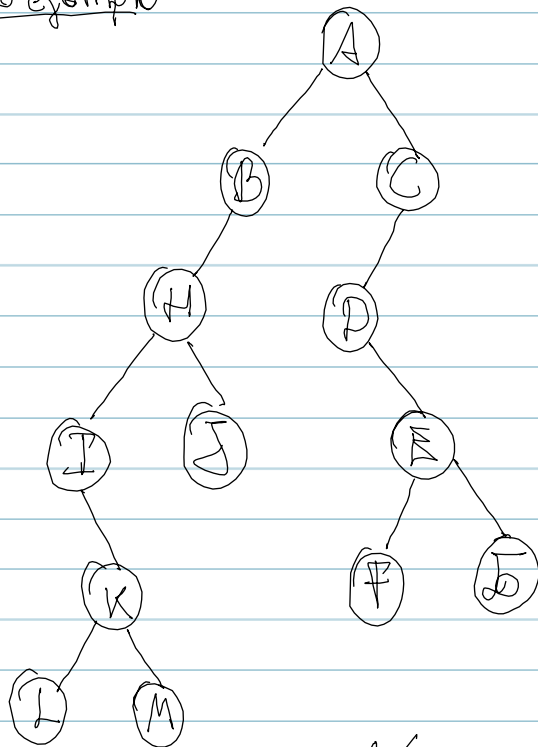
root = 15

L = {};

DepthFirstScan [T, root, ↓ "Postvisit Vertex" → (L = Append [L, #])
 Q) ↓];

Print [L] → {1, 3, 8, 13, 16, 6, 10, 14, 17, 18, 9, 5, 4, 7, 11, 2, 20, 19, 12, 15}

Otro ejemplo



Orden inicial
 $\hat{A} B H I K L M J C D E F G$

Orden final
 $\hat{L} M K I J H B F G E D C A$

Orden intermedio ✓
 $I L K M H J B A D F E G C$

$k=2$

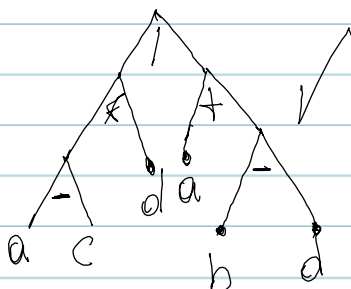
Definición Sea $T=(V,E)$ un árbol binario de una expresión algebraica A y h una lista formada por cada uno de los vértices de T entonces es:

1. Si h es el resultado del recorrido prefijo sobre T , se llama a h notación polaca de A .

2. Si h se elaboró a través de un recorrido postfijo sobre T se llama a h notación polaca inversa de A .

Ejemplo 1

$((a-c)*d)/(a+(b-d))$ ✓

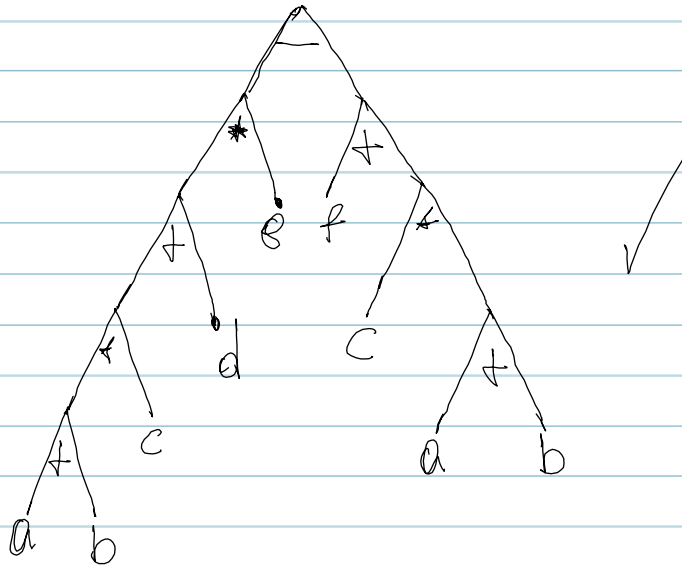


Notación polaca:
 $/ * - a c d + a - b d$

Notación polaca inversa:
 $a c - d * a b d - + /$

Example 2

$$((a+b) * c + d) * e - ((a+b) * c + f) \checkmark$$

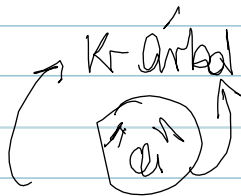


Pohars:

Pohaca:
 $\rightarrow \cancel{f} + \cancel{e} + abcde + f \cancel{c} + ab \checkmark$

Relaca inversa

Polaca inversa:
 $a b + c * d + e * f c a b + * + - \checkmark$



Árboles generadores

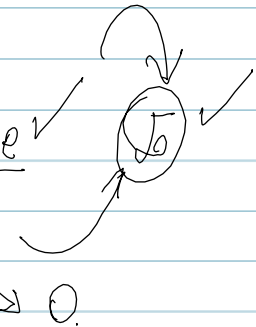
Definición Sea G un grafo no dirigido conexo, se llama árbol de expansión o generador de G , a un árbol subgrafo de G , que contiene todos sus nodos.

Teorema Sea G un grafo no dirigido, G tiene un árbol generador si y solo si G es conexo.

En Mathematica:

Shortest Path Spanning Tree ✓

Number Of Spanning Trees ✓



Ejemplo Construya en Mathematica una función que reciba como parámetro un grafo G y devuelva un árbol generador por cada vértice, retornando además, el número de árboles de expansión que posee.

Usando el software:

<< Combinatorica ✓

ABTG [G_Graph] := If [EdgeConnectivity [G] ≠ 0,

Module [d, La = Vertices [G], ListaGrafos = {}],

→ For [i = 1, i ≤ Length [La], ListaGrafos = Append [ListaGrafos,

ShortestPathSpanningTree [G, i]]]; i++];

Print [ShowGraphArray [ListaGrafos]];

Print ["El número de árboles de expansión del grafo es: ",

NumberOfSpanningTrees [G]]];

Print ["El grafo es desconexo no posee árboles generadores"]];

Existen dos algoritmos:

1. Buscar primero a lo ancho ✓
2. Buscar primero a lo largo ✓

Teorema (Buscar primero a lo ancho) Sea $G = (V, E)$ un grafo conexo con $V = \{v_1, v_2, \dots, v_n\}$ el conjunto de vértices de G en ese orden, entonces:

1. Considere el conjunto control $C = \{v_1\}$ y T el árbol generador a construir, inicializado con la raíz v_1 .

2. Para cada $w \in C$, añada una arista $[w, v_j]$ de G a T , con v_j un vértice de G en el orden de V , donde $[w, v_j]$ no forme un circuito en T . Si no se pueden agregar más lados a T , se terminó. T es un árbol de expansión del grafo G .

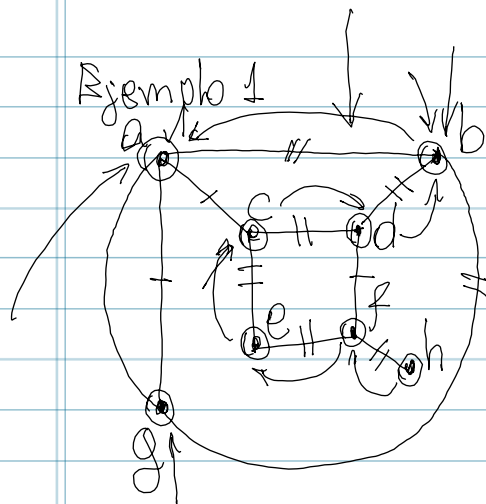
3. Sustituya los elementos del conjunto control C por los hijos en T de C , en el orden de V . Vaya al paso 2.

Teorema (Buscar primero a lo largo) Sea $G = (V, E)$ un grafo conexo con $V = \{v_1, v_2, \dots, v_n\}$ el conjunto de vértices de G en ese orden, entonces:

1. $w = v_1$ y T es el árbol generador a construir, inicializado con la raíz v_1 .

2. Añada una arista $[w, v_j]$ de G a T , con v_j un vértice de G en el orden de V , donde $[w, v_j]$ no forme un circuito en T . Actualice w como $w = v_j$ y repita este paso. Si no se pueden agregar más lados a T , vaya al paso siguiente.

3. Si $w = v_1$, se finalizó, T es un árbol generador del grafo G , sino, sea p el padre de w , tome a $w = p$ y vaya al paso 2.



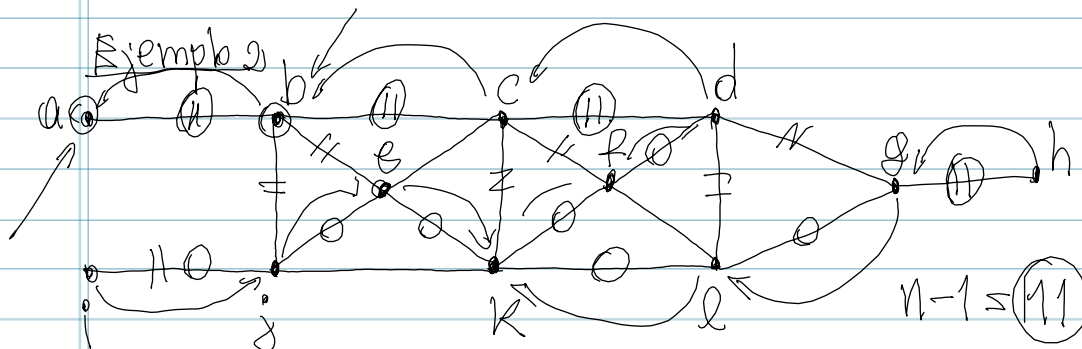
$$n-1 = 7$$

Buscar primero a lo ancho ✓

$$T = \{ [a, b], [a, c], [a, g], [b, d], [c, e], [d, f], [f, h] \}$$

Buscar primero a lo largo

$$T = \{ [a, b], [b, d], [d, c], [c, e], [e, f], [f, h], [b, g] \}$$



$$n-1 = 10$$

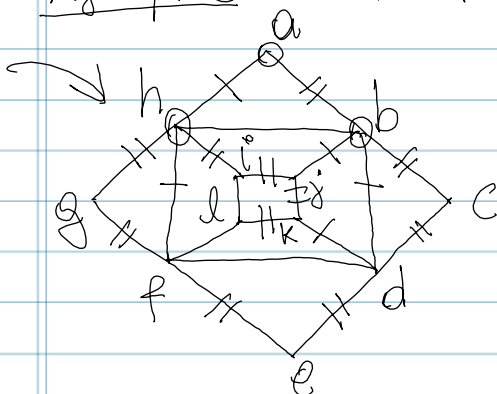
Buscar primero a lo ancho ✓

$$T = \{ [a, b], [b, c], [b, e], [b, j], [c, d], [c, f], [c, k], [j, i], [d, g], [d, l], [g, h] \}$$

Buscar primero a lo largo ✓

$$T = \{ [a, b], [b, c], [c, d], [d, f], [f, k], [k, e], [e, g], [j, i], [k, l], [l, g], [g, h] \}$$

Ejemplo 3 $n-1 = 11$



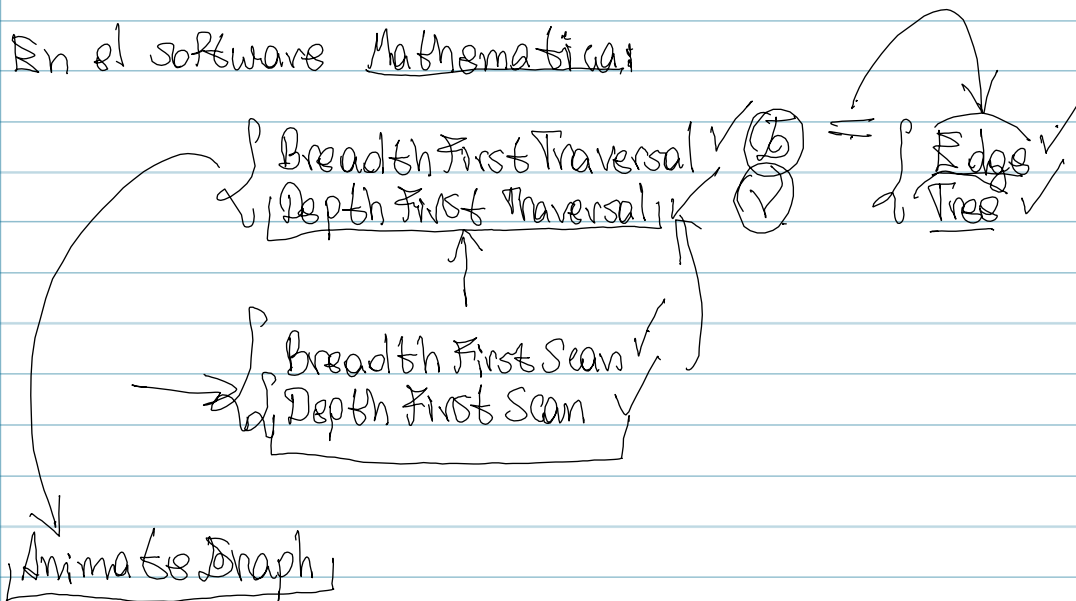
Buscar primero a lo ancho ✓

$$T = \{ [a, b], [a, h], [b, c], [b, d], [b, j], [h, f], [h, g], [h, i], [d, e], [d, k], [f, l] \}$$

Buscar primero a profundidad ✓

$$T = \{ [a, b], [b, c], [c, d], [d, e], [e, f], [f, g], [g, h], [h, i], [i, j], [j, k], [k, l] \}$$

En el software Mathematica:



Combinatorica

$n = 2$ 1, 2, 1, 6, 1, 1, 7, 1, 2, 3, 1, 3, 4, 1, 2, 4, 1, 5, 1, 1, 5, 6, 1, 5, 8, 1, 6, 4, 1, 7, 8;

show Graph

Set Graph Options [From Unordered Pairs [n],

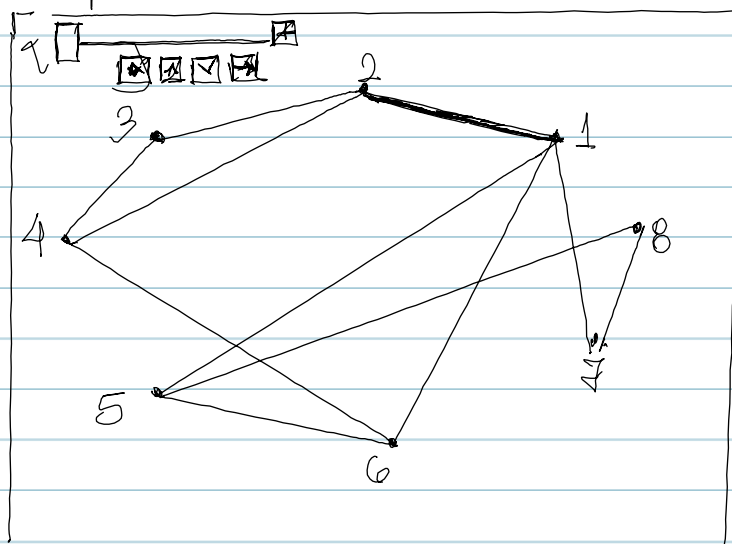
vertex Color → Black, Edge Color → Black],

vertex Number → True, Plot Range → 0.1];

Breadth First Traversal [1, Edge];

Animate Graph [1, 1, vertex Number → True, Plot Range → 0.1]

Quit



Ejemplo Elabore en Mathematica un programa que resuelva la búsqueda
a lo "largo" de un dato w sobre un k -árbol.

<< Combinatoria

```
→ h = 0;  
(k) = Input["Digite el orden del k-árbol: "];  
(n) = Input["Digite la cantidad de vértices del k-árbol: "];  
(w) = Input["Digite el dato de búsqueda: "];  
Show Graph [G = CompleteKaryFree [n, k], VertexNumber -> True,  
PlotRange -> 0.1]  
(v) = DepthFirstTraversal [G, 1];  
→ For [i = 1, i ≤ Length[v], & If [w == v[[i]], h = i; Break[]]; i++]  
→ If [h == 0, Print["Dato no encontrado"]]  
Print ["Dato encontrado"]]  
Quit[]
```

Árboles de expansión mínima

Definición Sea $G = (V, E)$ un grafo conexo y ponderado. Un árbol generador minimal de G es un árbol de expansión contenido en G , de peso mínimo. Se entiende el peso del árbol como la suma de los valores asignados a cada una de sus aristas en el grafo original.

En teoría de árboles:

1. Algoritmo de Prim
2. Algoritmo de Kruskal

Teorema (Algoritmo de Prim) Sea $G = (V, E)$ un grafo conexo y ponderado. Suponga que los elementos de $V = \{v_1, v_2, \dots, v_n\}$ tienen un orden y T es el árbol generador de peso mínimo a construir, entonces:

1. Sea T un árbol que contiene a v_1 y ninguna arista.
2. Si el número de elementos que contiene T es igual a $n-1$ aristas de E , se ha finalizado, T es un árbol de expansión mínima, su peso corresponde a la suma de los valores asociados a cada uno de sus lados.
3. Añada a T una arista de peso mínimo siempre que esta sea incidente a alguno de los vértices de T , no pertenezca a T y no forme un circuito en T . Si existe más de un lado $\{v_i, v_j\}$ de peso mínimo, elija el menor, sino, el j más pequeño. Vaya al paso 2.

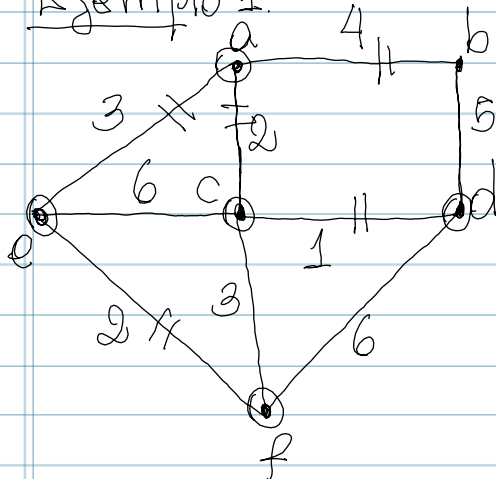
Teorema (Algoritmo de Kruskal) Sea $G = (V, E)$ un grafo conexo y ponderado. Suponga que T es el árbol generador de peso mínimo a construir, entonces:

1. Sea T un árbol que contiene todos los nodos de G y ningún lado.
2. Si T contiene $n-1$ elementos de E , se ha finalizado, T es un árbol de expansión mínima, su peso corresponde a la suma de los valores asignados a cada una de sus aristas.

③. Añada a T un lado de peso mínimo siempre que este no pertenezca a T y no forme un circuito en T . Si existe más de uno elija cualquier arista. Vaya al paso 2.

Kruskal puede devolver árboles de expansión mínimos distintos, mientras que Prim no.

Ejemplo 1. $n-1 = 5$



Prim:

$T = \{ [a,c], [c,d], [a,e], [e,f] \}$
 2 1 3 2

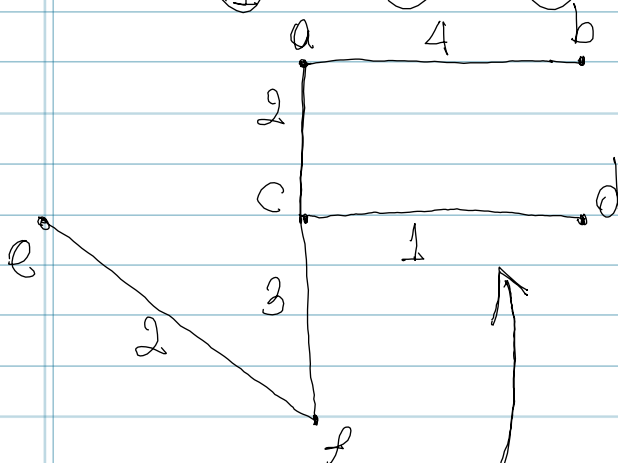
$[a,b] \nmid$, $Peso = 12$

$T = \{ [a,b], [a,c], [a,e], [c,d], [c,e], [c,f] \}$
 4 4 2 1 6 3

$[d,b], [d,f], [e,f] \nmid$
 5 6 2

Kruskal

$T = \{ [c,d], [a,c], [a,f], [c,e] \}$
 ① ② ② ③

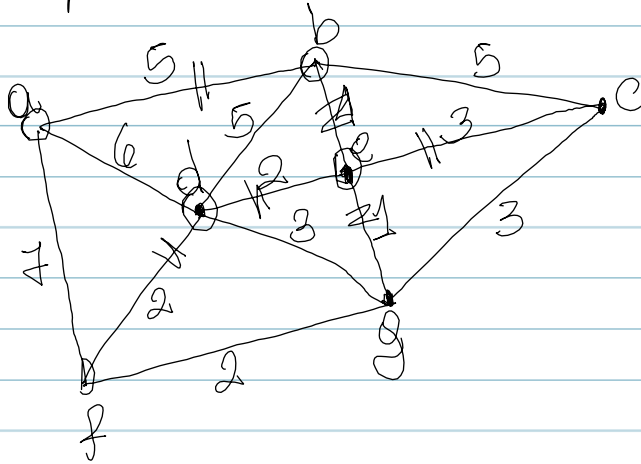


$[a,b] \nmid$
 ④

$Peso = 12$

Exemplo 2

$$n-1 = 6$$



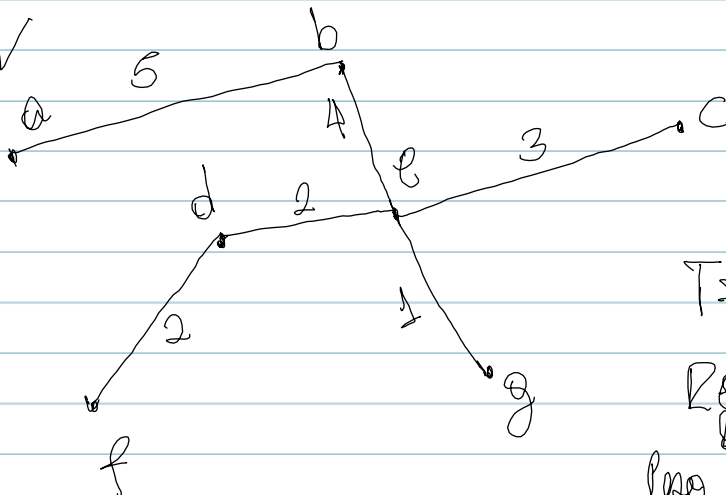
Prim ✓

$$T = \{ [a,b]_5, [b,e]_4, [e,g]_1, [e,d]_2, [d,f]_2, [e,c]_3 \}$$

$$P_{\text{max}} = 17$$

$$C = \{ [a/b]_5, [a/d]_6, [a/f]_7, [b,d]_5, [b,c]_5, [b/e]_4, [e,c]_3, [e/g]_1, [e/d]_2, [g,c]_3, [g,d]_3, [g/f]_2, [d/f]_2 \}$$

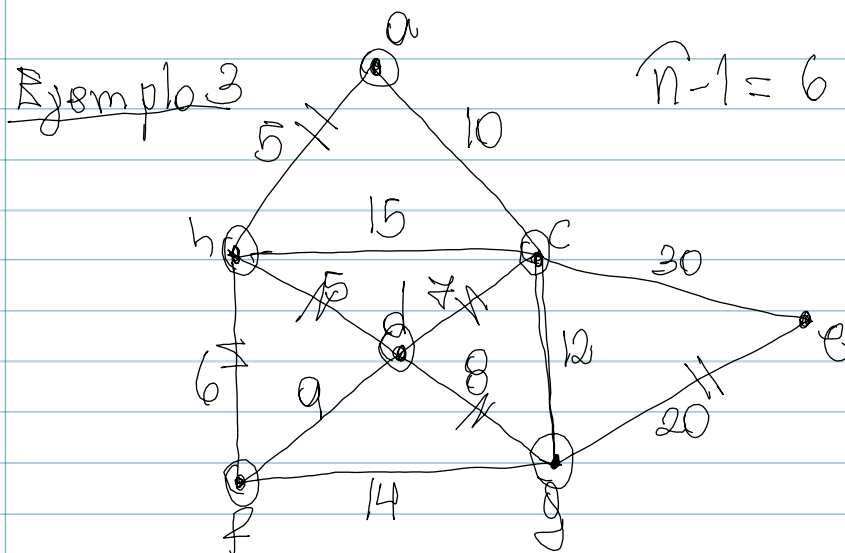
Kruskal ✓



$$T = \{ [e,g]_{(1)}, [d,f]_{(2)}, [d,e]_{(2)} \}$$

$$[e,c]_{(3)}, [b,e]_{(4)}, [a,b]_{(5)}$$

$$P_{\text{max}} = 17$$



Prim ✓

$T = \{ [a,b]_5, [b,d]_5, [b,f]_6, [d,c]_7, [d,g]_8, [g,e]_{20} \}$

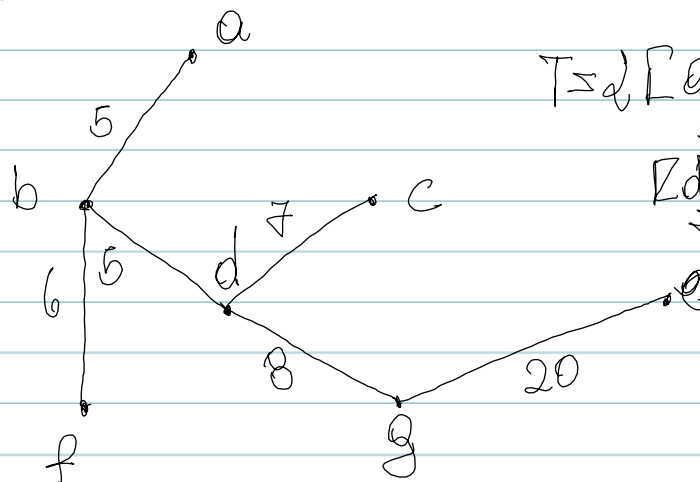
$P_{\text{now}} = 51$

$C = \{ [a/c]_{10}, [a/b]_5, [b/c]_{15}, [b/d]_5, [b/f]_6, [d/c]_7,$

$[d/g]_8, [d/f]_9, [f/g]_{14}, [c,e]_{30}, [e/g]_{12},$

$[g,e]_{20} \}$

Kruskal ✓



$T = \{ [a,b]_5, [b,d]_5, [b,f]_6,$
 $[d,c]_7, [d,g]_8, [g,e]_{20} \}$

$P_{\text{now}} = 51$

Una implementación del algoritmo de Prim en Mathematica:

Estructuras discretas con Mathematica

También:

Minimum Spanning Tree [5]

Una implementación:

<<Combinatorica

AlKruskal[G, Graph] :=

If[EdgeConnectivity[G] ≠ 0,

Module[{ArbolExpansion = MinimumSpanningTree[G];

Pesos = GetEdgeWeights[G, Edges[ArbolExpansion]];

pesominimo = Total[Pesos];

Print["Las aristas en orden de acuerdo con el algoritmo de
Kruskal del árbol de expansión de peso mínimo son: ", Edges[
ArbolExpansion]];]

Print["El peso mínimo es: ", pesominimo];

Print["El árbol de expansión minimal se muestra en la siguiente
animación: "];]

AnimateGraph[G, Edges[ArbolExpansion], VertexNumber -> True,
PlotRange -> 0.1]]

Print["El grafo no es conexo, el algoritmo de Kruskal no es aplicable"]]

También:

Maximum Spanning Tree