Teoría de árboles

Prof. Enrique Vílchez Quesada

Universidad Nacional de Costa Rica

Introducción

Los árboles son tipos especiales de grafos. Un uso común de estas representaciones, radica en asignar a cada uno de sus vértices un dato. Esto permite ordenar todos los datos ingresados, a través de la forma que adquiere el grafo. A esta aplicación se le denomina "estructura de datos" y será cubierta más adelante en el presente texto.
En la práctica hay múltiples ejemplos de árboles. El árbol genealógico de una familia forma un árbol en un sentido estricto dentro de esta teoría. El esquema de navegación en algunos sitios web, posee un diseño por "niveles de profundidad" con respecto a la página principal (el *index*), esta jerarquía define un árbol.

Introducción

También, las posibles estrategias a seguir en un juego, muchas veces se representan a través de árboles. Tal es el caso del popular juego "el gato" o "tres en raya", que probablemente el lector recordará de su infancia. En éste, un jugador debe completar alguna casilla vacía de nueve (en un inicio) con un cero o una equis según corresponda, de acuerdo con el símbolo escogido. En cada movida el objetivo es formar una fila, columna o diagonal con los mismos símbolos (ceros o equis). Si alguno de los dos jugadores lo consigue, gana la partida. Todas las posibles jugadas a realizarse, se pueden representar utilizando un árbol, tal y como se muestra en la figura 1.

Introducción

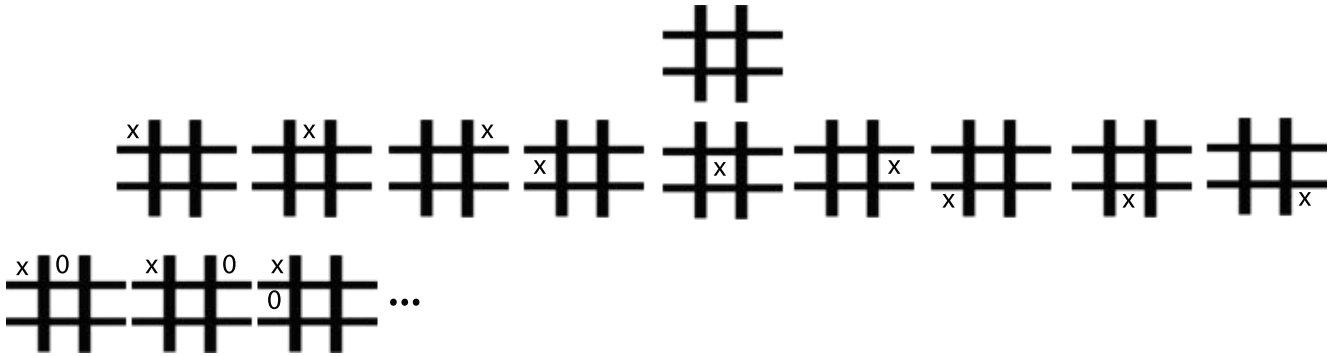


Figura: Juego “el gato” o “tres en raya”

Los árboles en general contienen un nodo de inicio llamado “raíz” y una serie de niveles que determinan una relación jerárquica entre los vértices. Consideremos la definición de árbol.

Definición 6.1

Definition (6.1)

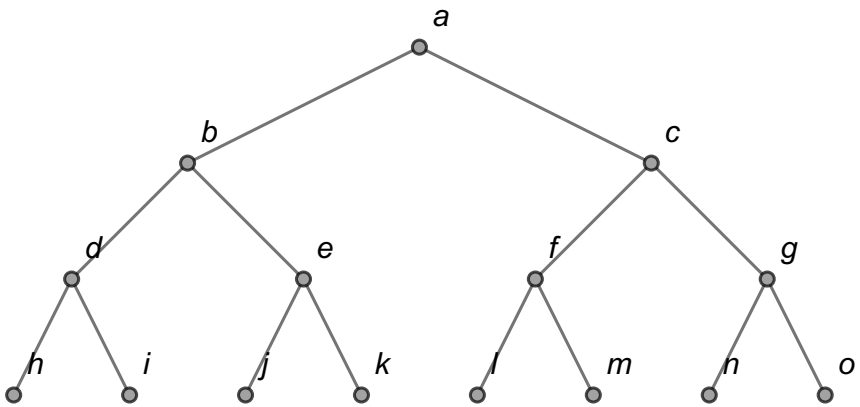
Un árbol (T, r) es un grafo $T = (V, E)$, donde para cualquier par de vértices distintos de V existe una única ruta que los une. Al nodo r se le llama raíz de T y se dice que se encuentra en el nivel 0 del árbol. Si $w_1 \to w_2 \to \cdots \to w_n$ es una trayectoria sobre T, con $r = w_1$, se define que el vértice w_{i-1} es padre de w_i, o bien, w_i es hijo de w_{i-1}. Los hijos de la raíz se encuentran en el nivel 1 de T, los hijos de los hijos de la raíz de T, si es que existen, se dice que están en el nivel 2 y así sucesivamente. Se define que w_i es un antecesor de w_j, si $i < j$, además, si esto ocurre, w_j se llama sucesor de w_i. Dos nodos que tienen el mismo padre se denominan hermanos.

Definición 6.1. Continuación

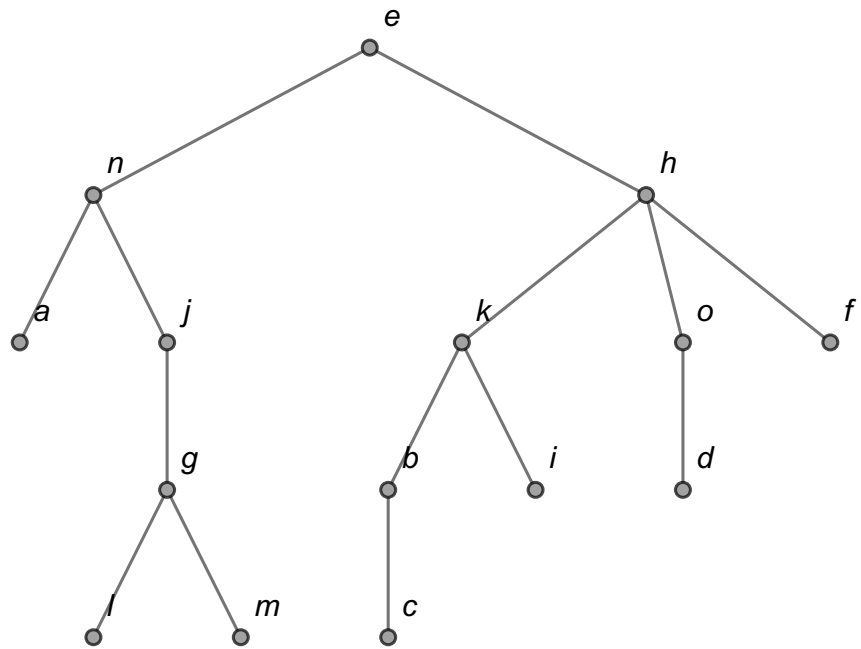
Los vértices en T que no son padres, se llaman nodos terminales, finales u hojas del árbol. Un vértice padre o no terminal es un nodo interno de T. Un $k-$árbol es aquel, donde cualquier padre tiene a lo sumo k hijos. Si en particular todo padre posee exactamente k hijos, el árbol se llama completo. Si $k = 2$ un $k-$árbol se denomina árbol binario. La altura h de un árbol T se define como el nivel máximo alcanzado. Un árbol T se dice que es balanceado o equilibrado si cualquier hoja se ubica en el nivel h, o bien, $h - 1$ y a su vez, cada padre tiene el mismo número de hijos exceptuando aquellos vértices no terminales que se encuentran en el nivel $h - 1$.

Comentario sobre la definición 1

La definición 1 indica de manera indirecta que un grafo T es un árbol sí y solo sí T es conexo y no contiene ningún circuito. Es importante señalar además, que la raíz de un árbol es el nodo antecesor de todos los vértices del grafo.
En la subfigura 2a se comparte un ejemplo de un árbol binario (de orden 2) con raíz a, de altura $h = 3$, completo y balanceado, cuyas hojas son h, i, j, k, l, m, n y o. En la subfigura 2b se presenta un ejemplo de un árbol de orden 3 con raíz e, de altura $h = 4$, no completo pues existen padres que poseen una cantidad de hijos menor a 3 y no balanceado, pues hay nodos terminales en un nivel distinto a $h = 4$, o bien, a $h - 1 = 3$, tal es el caso, de los vértices terminales a y f que se encuentran en el nivel 2 del árbol. Además, las hojas de este árbol corresponden a a, c, d, f, l, m e i.



(a) Árbol binario



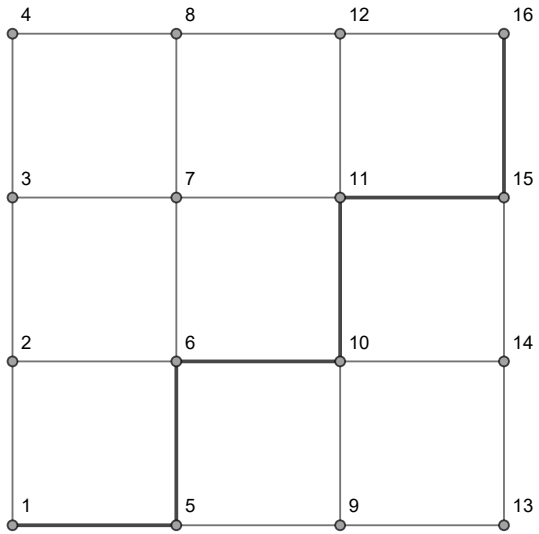
(b) Árbol de orden 3

Figura: Ejemplos de árboles

Los árboles suelen proveer mecanismos de resolución de problemas en distintas situaciones, veamos un ejemplo de ello.

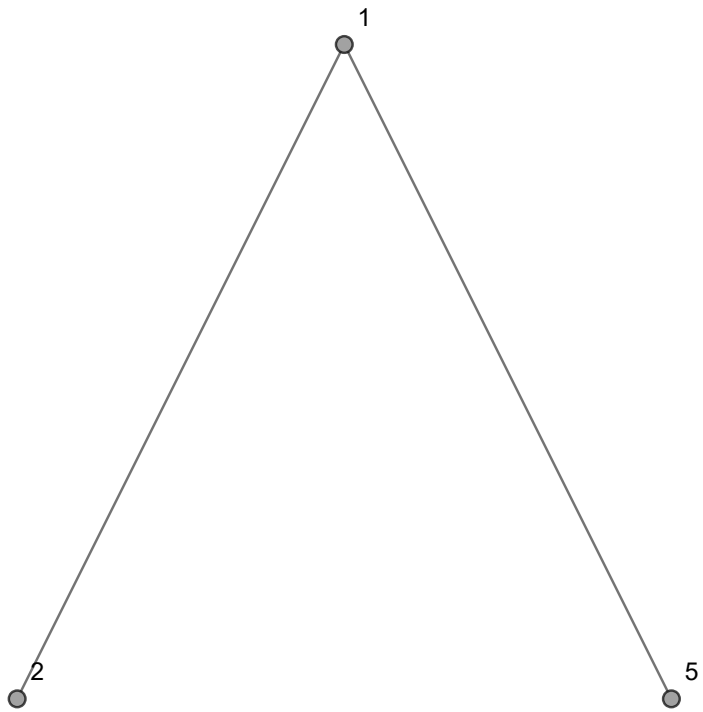
Example (6.1)

En el grafo red presentado a continuación, determine ¿cuántas trayectorias existen del nodo 1 al vértice 16, si en el camino recorrido se permite solo trasladarse horizontalmente hacia la derecha y verticalmente hacia arriba? En la siguiente figura se comparte un ejemplo de este tipo de rutas.



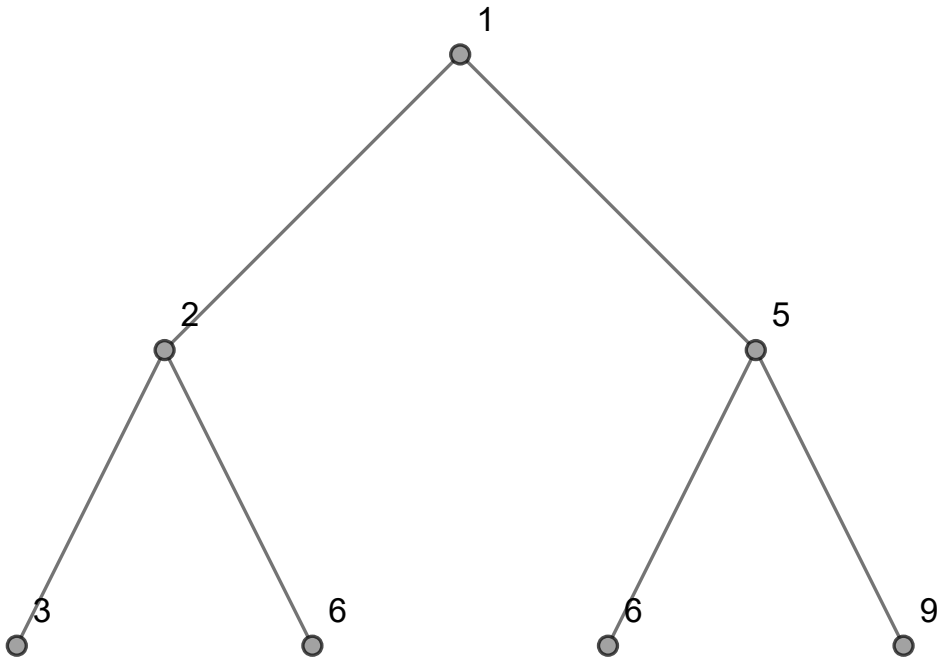
Solución del ejemplo 6.1

En este problema no se señala la necesidad de usar árboles, sin embargo, al comenzar a construir las trayectorias solicitadas se aprecia el empleo implícito de un árbol binario. Al iniciar en el vértice 1 es factible dirigirse al nodo 2, o bien, al vértice 5, lo cuál se representa así:



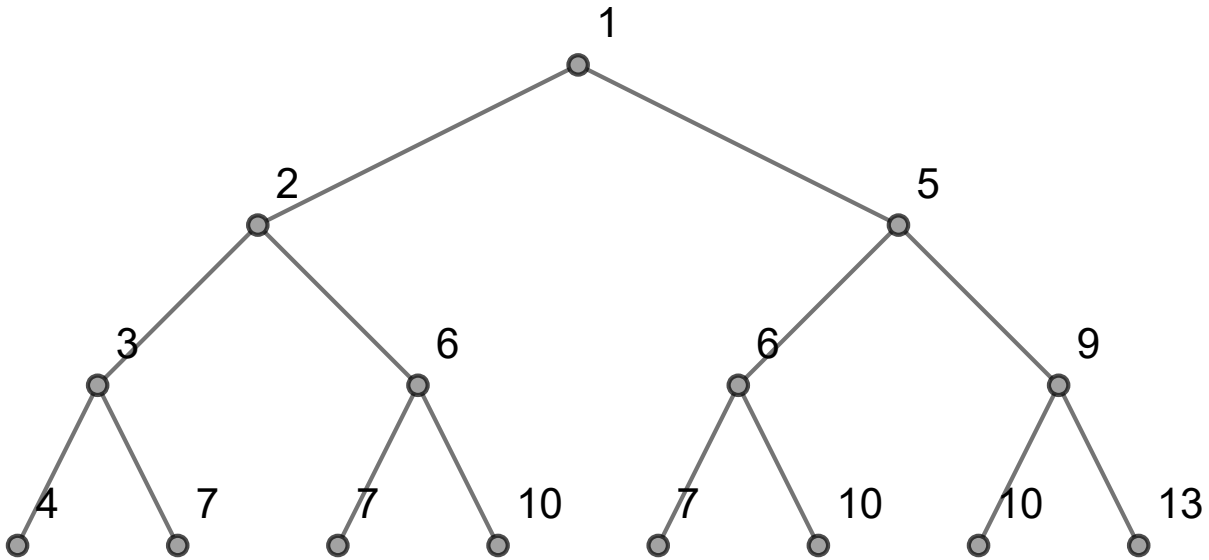
Solución del ejemplo 6.1

Al ubircarnos en 2, de allí la ruta puede proseguir al nodo 3, o bien, al vértice 6. En 5 es posible seleccionar el nodo 6 o el vértice 9. Esto se simboliza gráficamente como sigue:



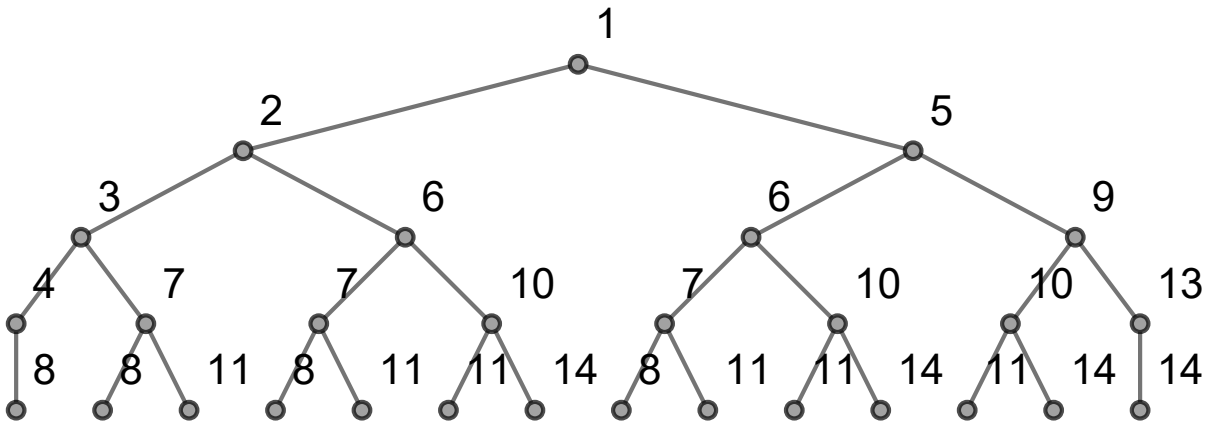
Solución del ejemplo 6.1

Ahora en 3 las reglas de construcción de la ruta admiten elegir 4 o 7. En 6 se puede seleccionar el vértice 7 o 10. En el nodo 9 es posible elegir 10 o 13. Formándose el árbol:



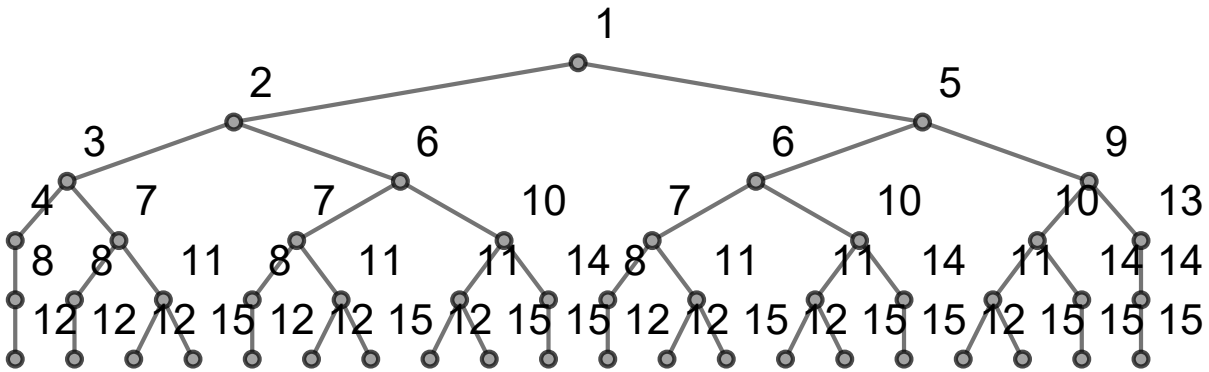
Solución del ejemplo 6.1

En 4 solo se permite seleccionar el nodo 8. En 7 se tiene la posibilidad de elegir 8 o 11. En 10 se escoge el nodo 11, o bien, el vértice 14. En 13 solo es factible elegir el nodo 14. Quedándo el árbol:



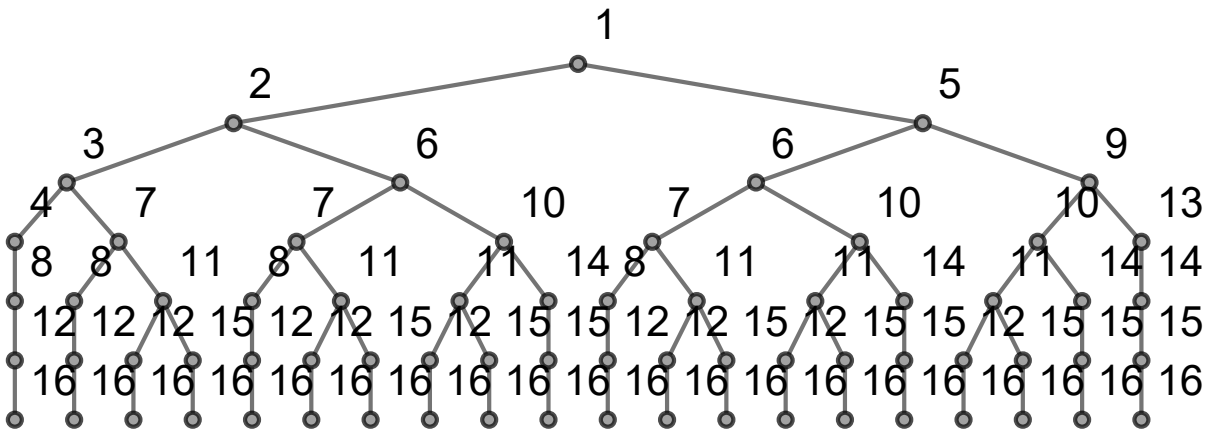
Solución del ejemplo 6.1

Ahora en 8 solo se puede seleccionar el vértice 12. En 11 se escoge el nodo 12, o bien, el vértice 15. En 14 solo se tiene disponible el nodo 15. Luego:



Solución del ejemplo 6.1

Finalmente, se completan todos los caminos de 1 a 16 bajo las reglas establecidas, donde de 12 y de 15 solo se acepta escoger el nodo 16 :



La cantidad de hojas igual a 20 de este último árbol binario corresponde al número de trayectorias buscado.

Nota

En un futuro curso de técnicas de conteo, el alumno estudiará una fórmula que resuelve directamente este problema, a saber:

$$\frac{(A+D)!}{A! \cdot D!}$$

Siendo A la cantidad de veces que se consigue ir hacia arriba en la cuadrícula y D el número de veces que se puede viajar a la derecha. En este ejercicio, $A = 3$ y $D = 3$, de donde:

$$\frac{(3+3)!}{3! \cdot 3!} = \frac{6!}{3! \cdot 3!} = 20$$

Descargue un archivo

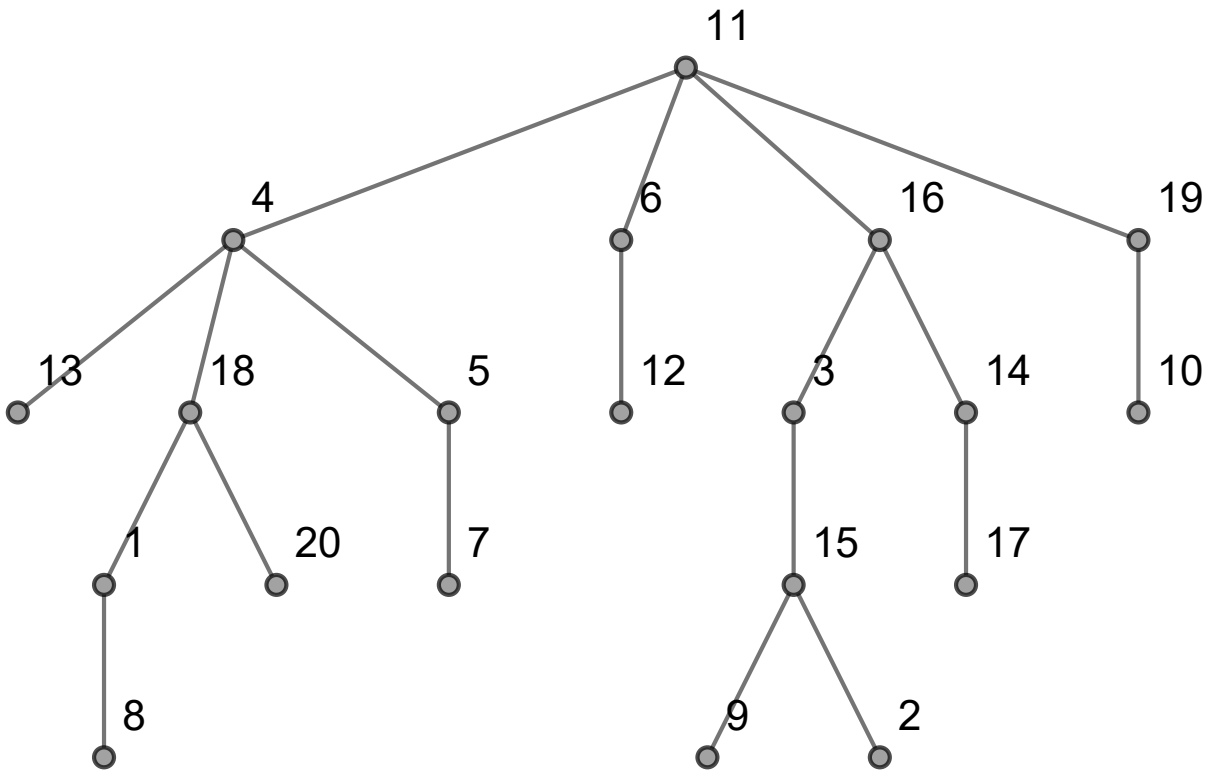
https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-135.zip

Comandos `Arbol` y `ArbolR`

En *Wolfram Mathematica* un árbol se crea por medio de las instrucciones del paquete **VilCretas**: `Arbol` o `ArbolR`. Ambas, reciben como argumento las aristas que conforman el grafo, al igual que lo hace el comando `Grafo` estudiado en el capítulo anterior. De hecho, la instrucción `Grafo` retorna un árbol cuando el grafo correspondiente es conexo y no contiene circuitos. La sentencia `ArbolR`, a diferencia del comando `Arbol`, requiere de un segundo argumento donde se especifica el nodo raíz. En `Arbol` el software *Mathematica* selecciona internamente la raíz, de acuerdo con el orden de los lados ingresados como parámetro.

Comandos `Arbol` y `ArbolR`

Por ejemplo, considere el siguiente árbol:



(1)

Comandos Arbol y ArbolR

En *Wolfram* éste se puede generar así:

In[ ] :=

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

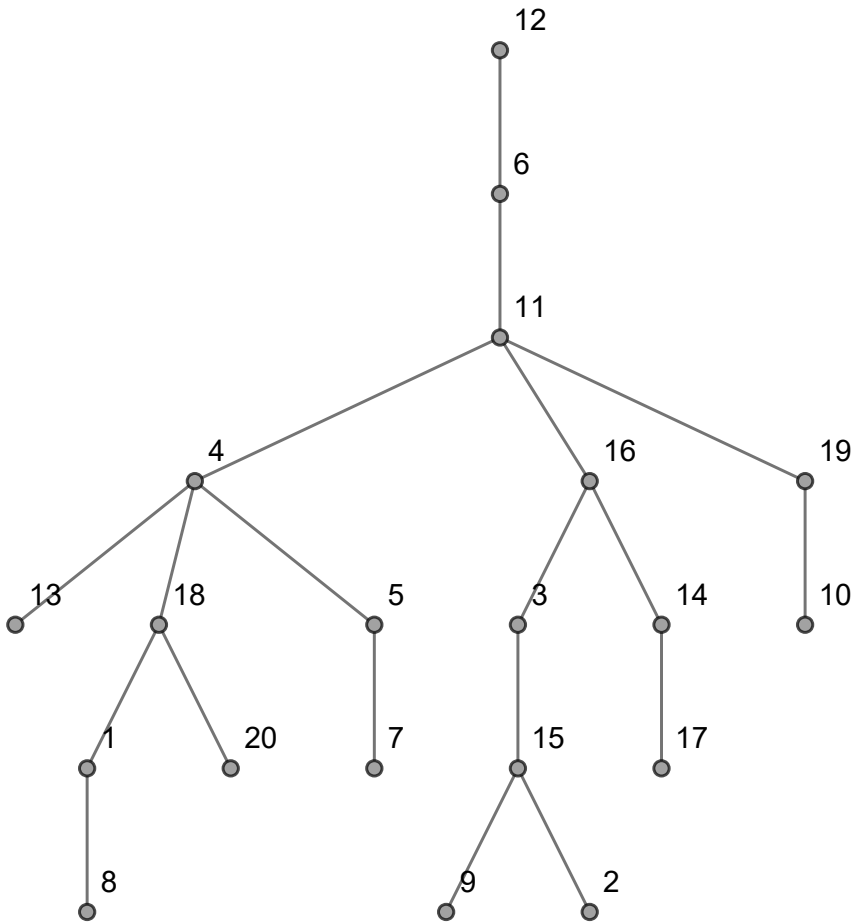
Si se asume ahora que la raíz del árbol 1 es el vértice 12, conviene emplear la instrucción `ArbolR`:

In[ ] :=

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

Comandos Arbol y ArbolR

Out[ ] =



Comandos `Arbol` y `ArbolR`

Tanto, `Arbol` como `ArbolR`, poseen el atributo `dirigido -> True` si desea construir un árbol dirigido. Se advierte al estudiante que en este libro, no se utilizarán árboles con aristas direccionadas.

Nota

`Arbol` y `ArbolR` corren sin dificultad, aún cuando los vértices del grafo sean letras del abecedario, o bien, una combinación entre caracteres y números.

Comandos Arbol y ArbolR

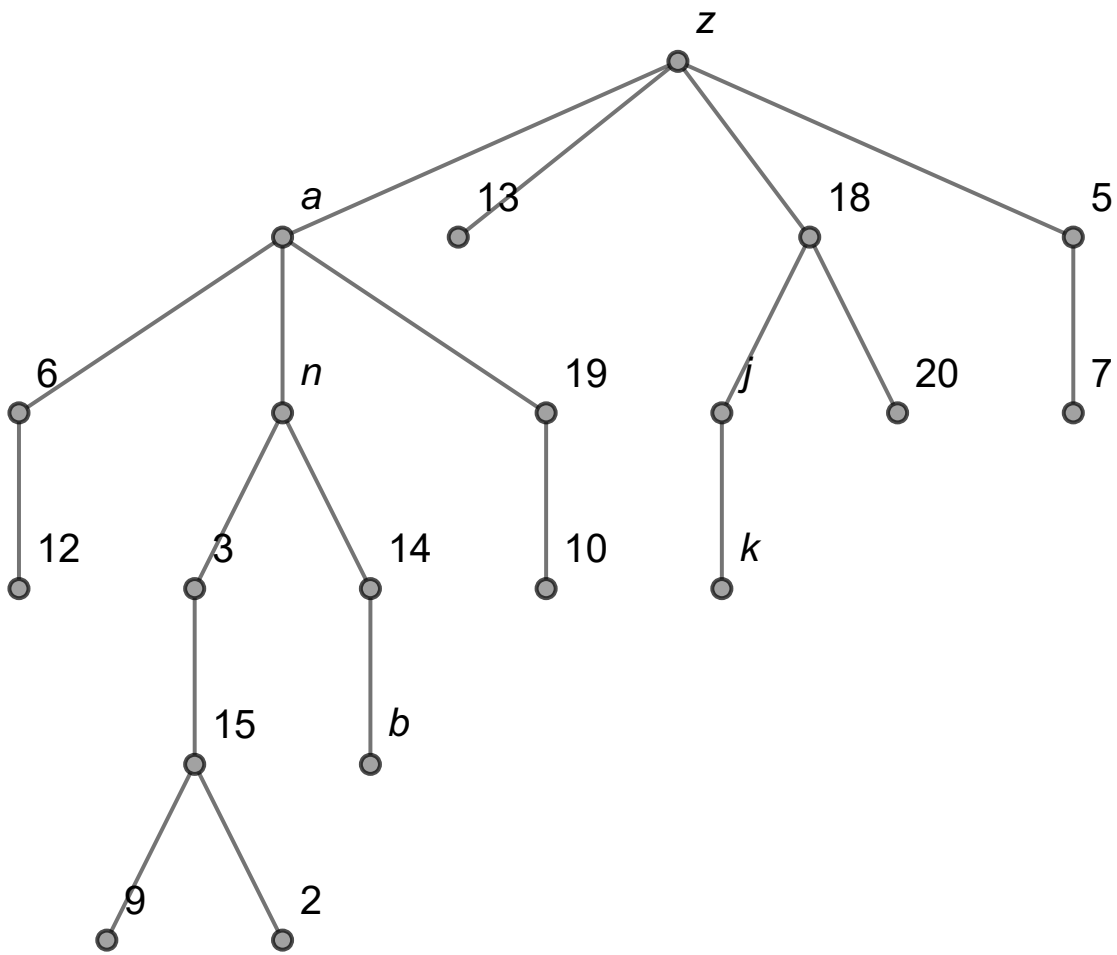
Por ejemplo:

In[ ] :=

```
ArbolR[{{a, z}, {a, 6}, {a, n}, {a, 19}, {z, 13}, {z, 18},
{z, 5}, {6, 12}, {n, 3}, {n, 14}, {19, 10}, {18, j},
{18, 20}, {5, 7}, {3, 15}, {14, b}, {j, k}, {15, 9},
{15, 2}}, z]
```

Comandos Arbol y ArbolR

Out[ ] =



Comandos `ArbolBinario` y `KArbol`

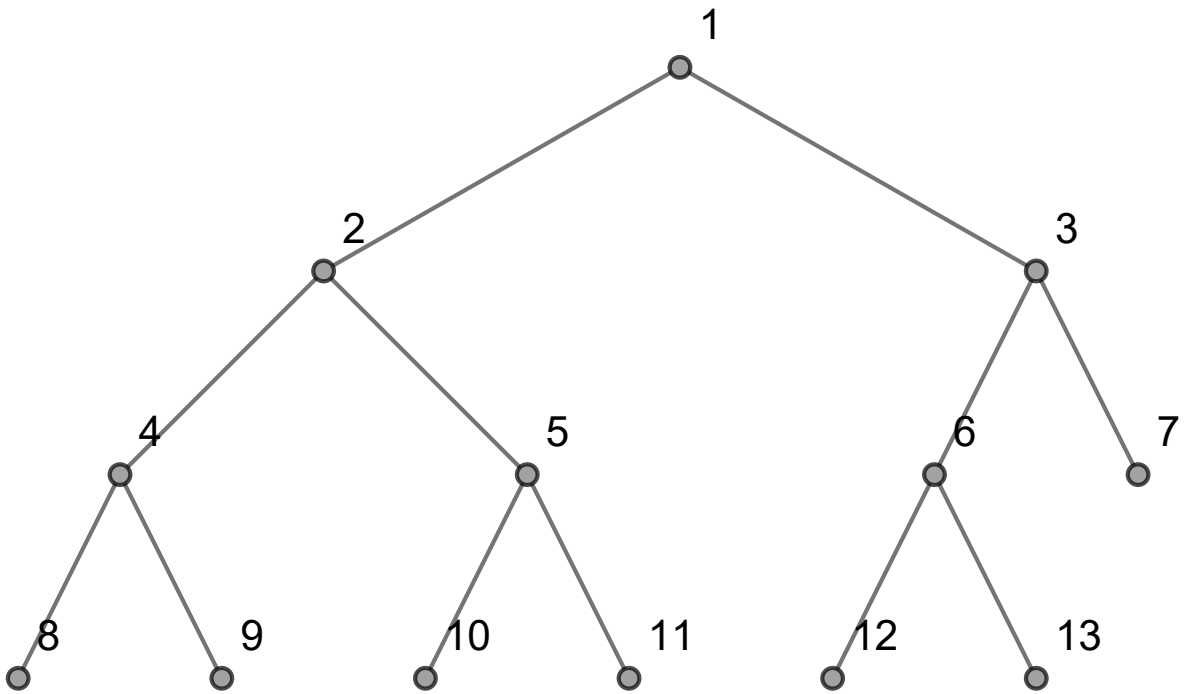
Por otra parte, el comando `ArbolBinario` de la librería **VilCretas** facilita el despliegue de un árbol binario con n nodos, siendo n un entero positivo pasado como argumento. Por ejemplo, la siguiente línea de código construye un árbol binario con 13 vértices:

In[ ] :=

```
ArbolBinario[13]
```

Comandos ArbolBinario y KArbol

Out[ ] =



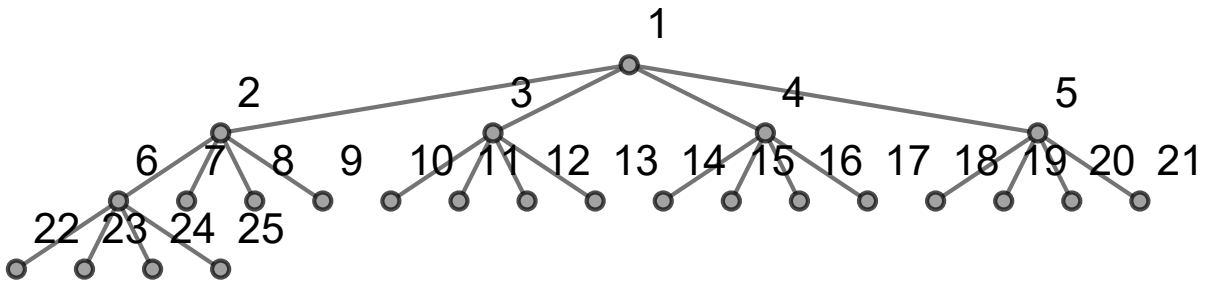
Comandos `ArbolBinario` y `KArbol`

También, la sentencia `KArbol` del paquete **VilCretas** permite elaborar un $k-$árbol con n nodos. El siguiente código construye un $4-$árbol con 25 vértices:

In[ ] :=

```
KArbol[4, 25]
```

Out[ ] =



Comandos `ArbolRandom` y `ArbolBinarioRandom`

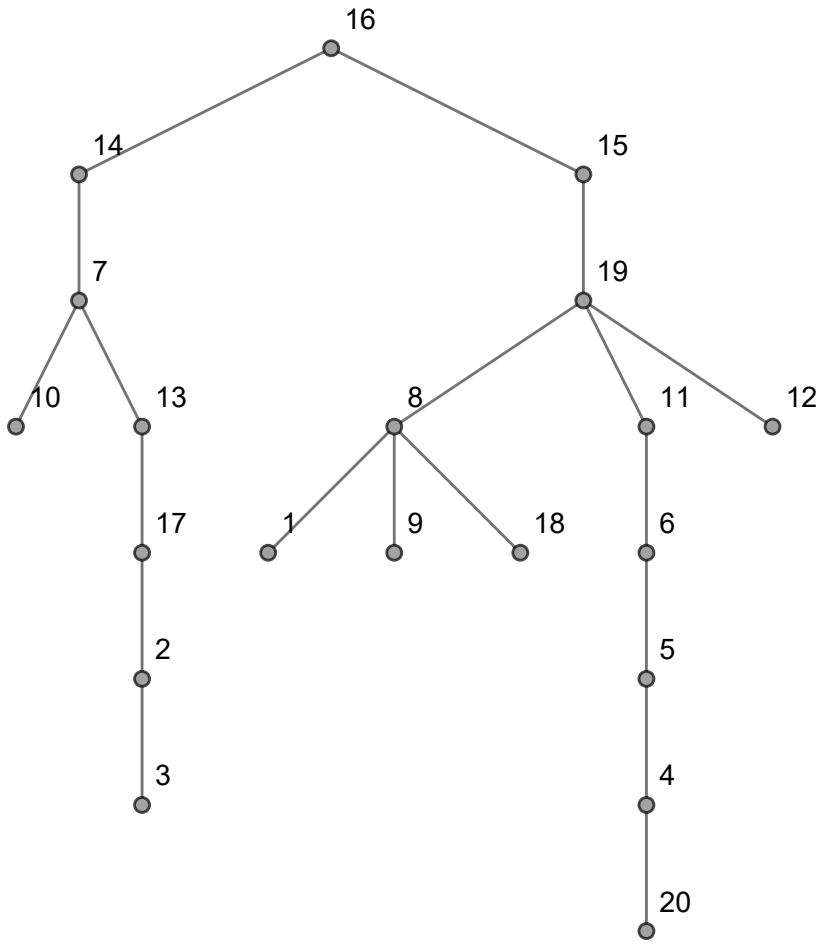
La librería **VilCretas** integra, además, algunas instrucciones que generan árboles pseudoaleatorios: `ArbolRandom` y `ArbolBinarioRandom`. Ambas, devuelven un árbol pseudoaleatorio con n vértices, con la diferencia de que `ArbolBinarioRandom`, como su nombre lo indica, retorna un árbol pseudoaleatorio exclusivamente binario. El siguiente código muestra un árbol pseudoaleatorio y un árbol pseudoaleatorio binario con 20 nodos:

In[ ] :=

```
ArbolRandom[20]
ArbolBinarioRandom[20]
```

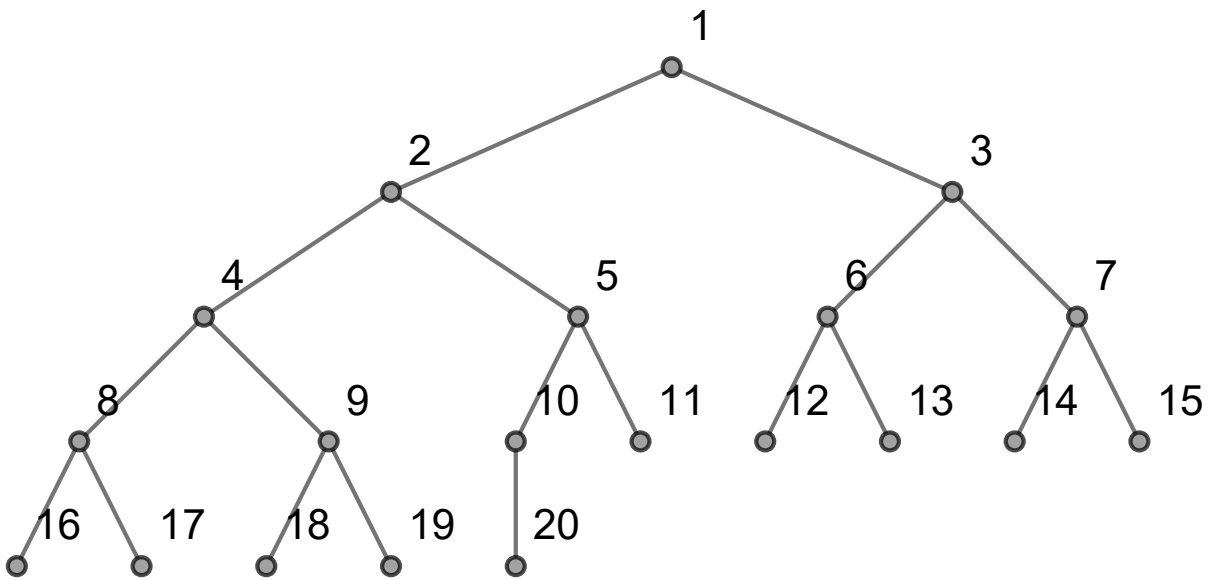
Comandos ArbolRandom y ArbolBinarioRandom

Out[ ] =



(2)

Comandos `ArbolRandom` y `ArbolBinarioRandom`



La salida es pseudoaleatoria por lo que posiblemente al ejecutar el mismo **In[ ]** en el ordenador del alumno, el resultado devuelto por el software será distinto.

Comandos `ArbolRandom` y `ArbolBinarioRandom`

Nota

Los comandos `ArbolBinario`, `KArbol`, `ArbolRandom` y `ArbolBinarioRandom` reciben como parámetro la cantidad n de vértices del árbol a crear. En todas estas instrucciones, por defecto *Mathematica* asume que los nodos son números enteros consecutivos iniciando en 1 y hasta n.

Uso de las instrucciones Arbol, ArbolR, ArbolBinario, KArbol, ArbolRandom y ArbolBinarioRandom

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-136.zip

CDF: construcción de un $k-$árbol

A continuación, además, se comparte la descarga de un documento con un formato computable que facilita la visualización de un $k-$árbol donde el usuario especifica el orden k, la cantidad de vértices del árbol y su raíz, retornándose el árbol correspondiente, sus hojas y su altura.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/CDFs/KArbol.cdf.zip

Comandos Hojas y Altura

Finalmente, el paquete **VilCretas** cuenta también, con los comandos `Hojas` y `Altura`, que determinan los nodos terminales y la altura de un árbol pasado como parámetro, respectivamente. Al usar estas sentencias sobre el árbol 2:

In[ ] :=

```
arbol = Arbol[{{1, 8}, {2, 3}, {2, 17}, {4, 5}, {4, 20},
{5, 6}, {6, 11}, {7, 10}, {7, 13}, {7, 14}, {8, 9},
{8, 18}, {8, 19}, {11, 19}, {12, 19}, {13, 17}, {14, 16},
{15, 16}, {15, 19}}];
Hojas[arbol]
Altura[arbol, 16]
```

Out[ ] =

```
{1, 3, 20, 10, 9, 18, 12}
7
```

Comandos `Hojas` y `Altura`

El estudiante debe notar que la instrucción `Altura` demanda indicar cuál es el vértice raíz del árbol. Además, la sentencia `Hojas` goza de la opción `raiz -> nodo` con la funcionalidad de especificar en `nodo` el vértice raíz, aspecto necesario solamente si la valencia del nodo raíz es igual a 1. En el ejemplo anterior, no se utilizó el atributo `raiz -> nodo` pues la raíz 16 del árbol correspondiente tiene grado 2. Si se omite el empleo de `raiz -> nodo` aún cuando la raíz posea grado igual a 1, *Mathematica* incluirá en la lista de vértices terminales dicho nodo.

Empleo de las sentencias Hojas y Altura

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-137.zip

Teorema 6.1

La definición 1 como ya se comentó, señala que un grafo es un árbol si éste es conexo y no contiene ningún circuito. El teorema que prosigue lo formaliza añadiendo dos resultados adicionales.

Theorem (6.1)

Sea $T = (V, E)$ un grafo con n vértices. T es un árbol sí y solo sí 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.*

Comentario sobre el teorema 3

El teorema 3 propone que todo árbol contiene $n-1$ aristas siendo n el número de nodos del grafo. Esta propiedad es esencial y se recurrirá a ella en los algoritmos de elaboración de cierto tipo de árboles llamados "árboles generadores" o de "expansión", tema que será cubierto más adelante. Por lo pronto, el estudiante debe tener claro, que todo árbol siempre poseerá una cantidad de lados igual a $n-1$.
Además, la librería **VilCretas** suministra el comando `ArbolJQ` que consiste en una aplicación directa de las propiedades enunciadas en el teorema 3. `ArbolJQ` devuelve como salida True, si el argumento recibido es un árbol y en caso contrario, False. Si el parámetro pasado a `ArbolJQ` es un grafo que no corresponde a un árbol, la sentencia ofrece una justificación del False, recurriendo a las propiedades del teorema 3.

Comentario sobre el teorema 3

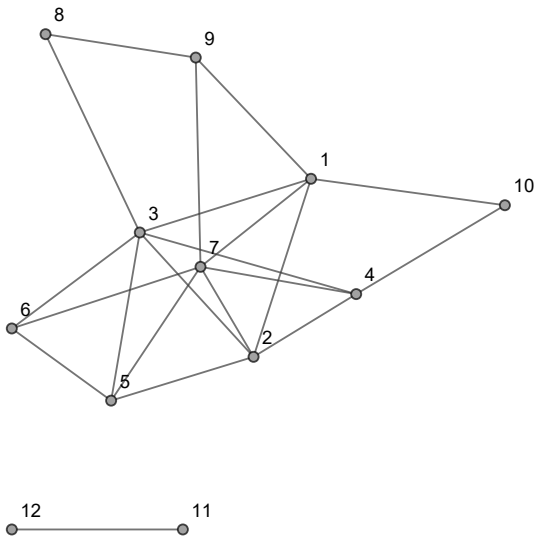
Consideremos el siguiente ejemplo:

In[ ] :=

```
grafo = Grafo[{{1, 2}, {1, 3}, {1, 7}, {1, 9}, {1, 10},
{2, 3}, {2, 4}, {2, 5}, {2, 7}, {3, 4}, {3, 5}, {3, 6},
{3, 8}, {4, 7}, {4, 10}, {5, 6}, {5, 7}, {6, 7}, {7, 9},
{8, 9}, {11, 12}}]
ArbolJQ[grafo]
```

Comentario sobre el teorema 3

Out[ ] =



False
No es conexo
Tiene circuitos
Se concluye que el grafo no es un árbol al ser disconexo y contener circuitos.

Uso de la sentencia `ArbolJQ`

Descargue un archivo

`https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-138.zip`

Teorema 6.2

Los árboles binarios son muy importantes dentro de esta teoría al tener diversas aplicaciones de empleo. Algunas de ellas serán abordadas en este texto. Iniciaremos su estudio con una primera propiedad. Cuando un árbol binario es completo, conociendo el número de nodos padres es posible obtener la cantidad de hojas y de vértices en total del árbol. El teorema próximo da garantía de esta característica.

Theorem (6.2)

Sea (T, r) un árbol binario completo con x vértices internos entonces T contiene $x + 1$ hojas y $2x + 1$ nodos en total.

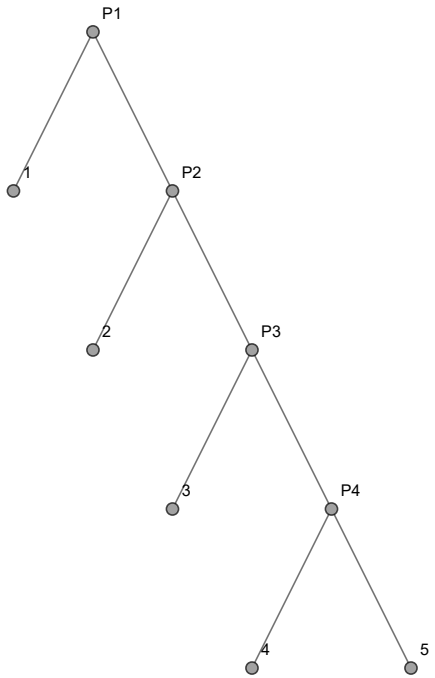
Veamos un ejemplo.

Example (6.2)

Si hay 100 equipos de fútbol en un torneo de eliminación simple, ¿cuántos partidos habrá para tener un ganador? Un torneo de eliminación simple es aquel donde juegan dos equipos y el vencedor continua en la competencia.

Solución del ejemplo 6.2

El torneo de eliminación simple descrito en el problema es factible representarlo mediante un árbol binario completo. Supongamos momentáneamente que el campeonato de fútbol tiene solo 5 equipos participantes, en dicho caso, un árbol que representa toda la competición es:



(3)

Solución del ejemplo 6.2

En el árbol se infiere que son necesarios 4 partidos para tener un campeón. Lo importante del grafo 3 es observar que el número de partidos corresponde a la cantidad de vértices internos.
Si x es el número de padres que contiene un árbol binario completo que simboliza el campeonato de fútbol con 100 equipos entonces este valor, queda determinado por lo enunciado en el teorema 4. Sabemos que el número de hojas de dicho árbol es igual a 100 (número de equipos) y por lo tanto, 100 tiene que ser igual a $x+1$ (número de vértices términales) de acuerdo con el teorema 4, luego:

$$x+1 = 100 \Rightarrow x = 99$$

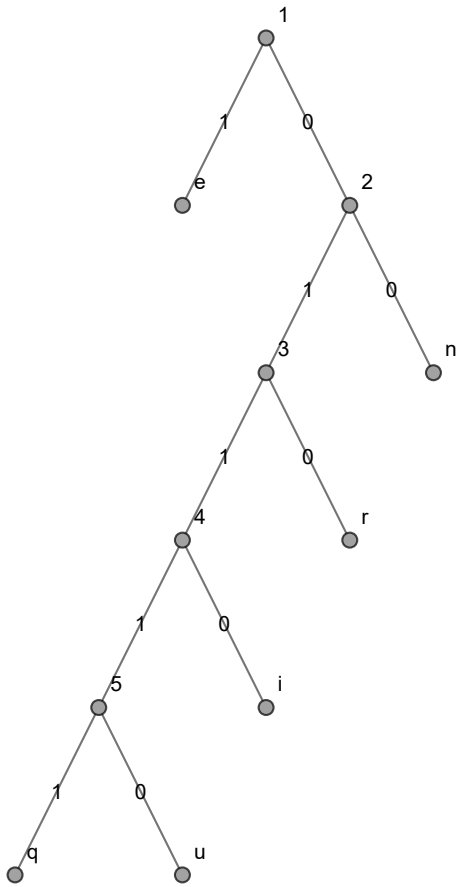
Se concluye que son necesarios 99 partidos para finalizar el torneo.

Códigos de Huffman

Una aplicación muy interesante de los árboles binarios completos, la conforma los códigos de *Huffman*. *David A. Huffman* siendo estudiante de doctorado en el *Instituto Tecnológico de Massachusetts* (*MIT*), creó este método para estructurar un sistema de codificación de caracteres de longitud variable. Si el lector desea mayor información sobre *David A. Huffman* se recomienda consultar *IEEE*.
Los sistemas de codificación normalmente usan longitudes fijas para representar cada caracter, tal es el caso del método *ASCII*. El aporte principal de los códigos de *Huffman*, reside en utilizar números binarios de distinta longitud para la creación de un abecedario. Las letras más utilizadas tendrán una longitud menor en comparación con las letras de menor frecuencia.

Códigos de Huffman

Por ejemplo, el siguiente árbol binario completo representa un sistema de codificación para la palabra "enrique":



(4)

Códigos de Huffman

Las hojas del árbol son las letras del abecedario A de interés, $A = \{$e, i, n, q, r, u$\}$. En la palabra "enrique", el caracter "e" tiene una frecuencia igual a 2 pues aparece dos veces, mientras que las otras letras al mostrarse una vez, poseen una frecuencia igual a 1. De allí que en este ejemplo, la "e" se exhibe con el menor nivel de profundidad con respecto a las otros caracteres en el árbol 4 de códigos de *Huffman*. Los lados del árbol se han etiquetado con un valor binario 1 si la dirección es a la izquierda y 0 en la dirección derecha. En otros textos, podría ocurrir que se defina la convención de colocar 1 en las aristas con dirección derecha y 0 en los lados con dirección izquierda.

Códigos de Huffman

De esta forma, la palabra "enrique" queda codificada como:

$$\underbrace{1}_{e}\ \underbrace{00}_{n}\ \underbrace{010}_{r}\ \underbrace{0110}_{¡}\ \underbrace{01111}_{q}\ \underbrace{01110}_{u}\ \underbrace{1}_{e}$$

Es decir: 10001001100111011101.
Además, dado un árbol de códigos de *Huffman* su uso podría consistir no solo en codificar una palabra, sino también, en decodificar una secuencia de *bits*.

Códigos de Huffman

Por ejemplo, empleando el árbol 4, si desea decodificar el mensaje contenido en el número binario 010011000, se comienza en la raíz del árbol y se inicia un recorrido en función de los *bits* alojados en el binario. Al transitar 010 se llega a la letra "r", siendo ésta el primer caracter del mensaje. Luego, se regresa a la raíz y se continua la lectura del binario en los dígitos posteriores a 010. Al recorrer los cuatro dígitos siguientes 0110 se obtiene la letra "i". Se regresa a la raíz leyendo el binario en los dígitos sucesores a 0110. Finalmente, 00 forma el caracter "n" y se infiere que la cadena de *bits* 010011000 corresponde a la palabra "rin" en el árbol 4 de códigos de *Huffman*.

Códigos de Huffman

Huffman también desarrolló un algoritmo que optimiza el árbol de codificación. Como se observa en 4, el árbol de codificación así construido de la palabra "enrique" es poco equilibrado. En un árbol como ese, al realizar en un ordenador una búsqueda u otros tipos de operaciones de "estructura de datos", como por ejemplo: una inserción o una eliminación de un dato, se consumirá mayor cantidad de recursos computacionales y a razón de ello, el tiempo de ejecución será mayor. Bajo esta perspectiva, al balancear un árbol que corresponde a una "estructura de datos", éste se optimiza.

- El teorema compartido a continuación, enuncia un algoritmo para construir un árbol de códigos de *Huffman* optimizado. Se aclara que en algunas ocasiones la implementación de este procedimiento no conduce a un árbol totalmente equilibrado, pero aún así, lo mejora.

Teorema 6.3. Algoritmo de códigos de *Huffman* optimizado

Theorem (6.3)

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

1. *Sean* f_i *y* f_j *las dos frecuencias menores de* L*, tómese a* $nf = f_i + f_j$ *y sustituya en* L*,* f_i *y* f_j *por* nf*. Se forma un árbol binario con raíz* nf *e hijos* a *y* b*, correspondientes a las frecuencias* f_i *y* 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*.*

Comentario sobre el teorema 6

Una variante de empleo del algoritmo de códigos de *Huffman* optimizado reside en asumir que las frecuencias de los caracteres del abecedario A de interés son todas distintas. Naturalmente, en la práctica ocurrirán algunas veces, empates en el número de apariciones de ciertas letras en una palabra. Si esto sucede se deben reasignar las frecuencias evitando los empates y respetando el orden de los caracteres, de tal forma que aquellos con mayor cantidad de repeticiones deberán poseer una frecuencia reasignada mayor correspondiente a su lugar de prioridad. En este libro se recurre a esta variante al utilizar el teorema 6.

Consideremos algunos ejercicios al respecto.

Example (6.3)

Elabore un árbol de códigos de *Huffman* optimizado para la palabra "enrique".

Solución del ejemplo 6.3

Se inicia extrayendo las frecuencias de cada uno de los caracteres del abecedario $A = \{\text{e, i, n, q, r, u}\}$:

Caracter	Frecuencia
e	2
n	1
r	1
i	1
q	1
u	1

Solución del ejemplo 6.3

Antes de emplear el teorema 6, se reasignarán las frecuencias de las letras por cuanto ocurren empates en algunas de ellas, lo anterior como una variante de uso de este algoritmo. Dado que la mayor frecuencia se da en la letra "e", se asignarán enteros consecutivos comenzando en 1 con el caracter al cual se le desea clasificar como el de menor frecuencia y finalizando en "e", al ser la letra de mayor nivel de importancia. Una opción sería:

Caracter	**Frecuencia reasignada**
u	1
q	2
i	3
r	4
n	5
e	6

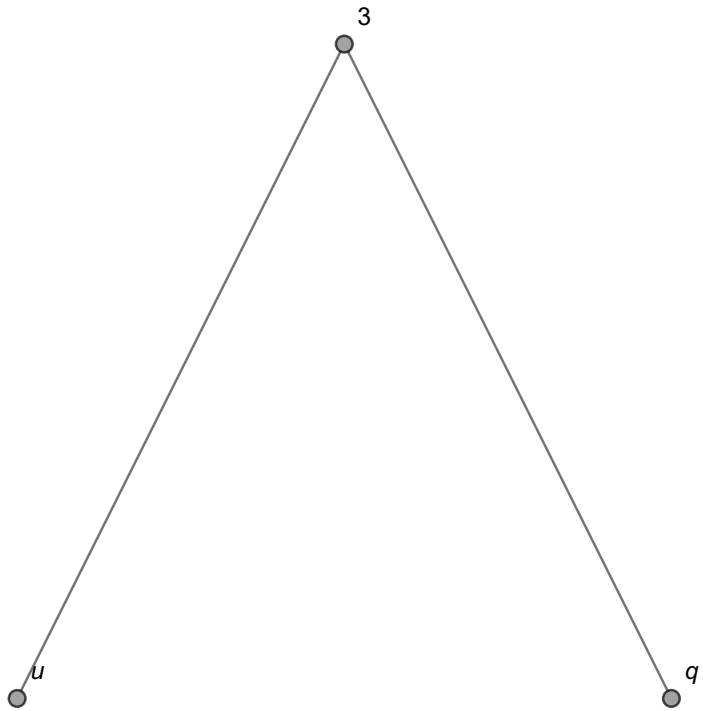
(5)

Solución del ejemplo 6.3

El estudiante debe tener claro que la reasignación de frecuencias tiene varias posibilidades en este ejemplo, siempre y cuando, se conserve la "e" con la mayor de las frecuencias asociadas.
De acuerdo con la tabla 5, la lista L de frecuencias de cada una de las letras que constituyen la palabra "enrique" corresponde a $L = \{1, 2, 3, 4, 5, 6\}$. Suponga que cada frecuencia se simboliza como f_i siendo i la posición del número en L.

Solución del ejemplo 6.3

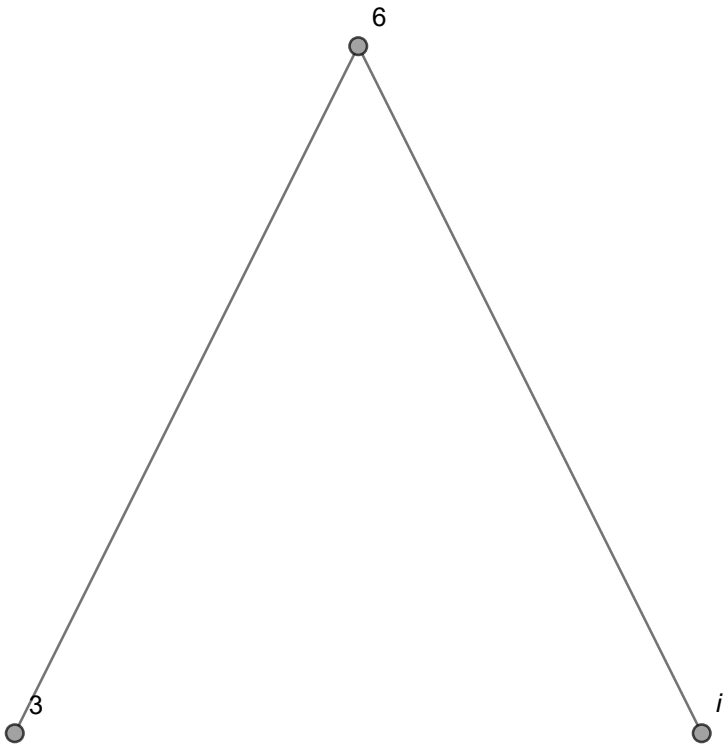
Ahora, por el teorema 6, se escogen las dos frecuencias menores $f_1 = 1$ y $f_2 = 2$, al sumarlas se obtiene $nf = 1 + 2 = 3$ y se reemplazan $f_1 = 1$ y $f_2 = 2$ por nf en L, es decir, $L = \{3, 3, 4, 5, 6\}$. Además, como $f_1 = 1$ es la frecuencia de "u" y $f_2 = 2$ es la frecuencia de "q", se forma el árbol:



(6)

Solución del ejemplo 6.3

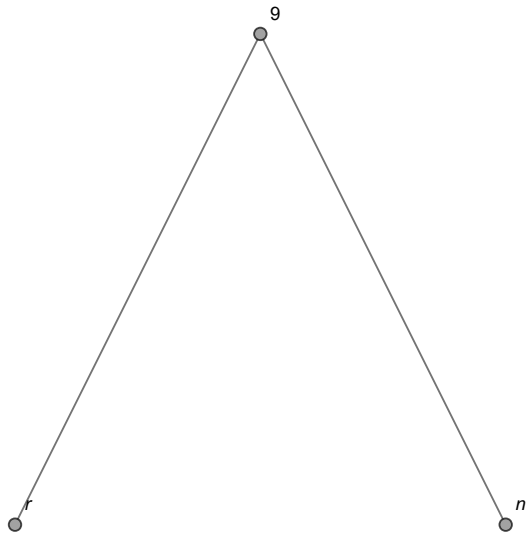
Luego, al repetir el proceso, se seleccionan de L las dos frecuencias menores $f_1 = 3$ y $f_2 = 3$, por lo que, $nf = 3 + 3 = 6$ y L se actualiza como: $L = \{6, 4, 5, 6\}$. También, al vincularse $f_2 = 3$ con la letra "i", se forma el árbol:



(7)

Solución del ejemplo 6.3

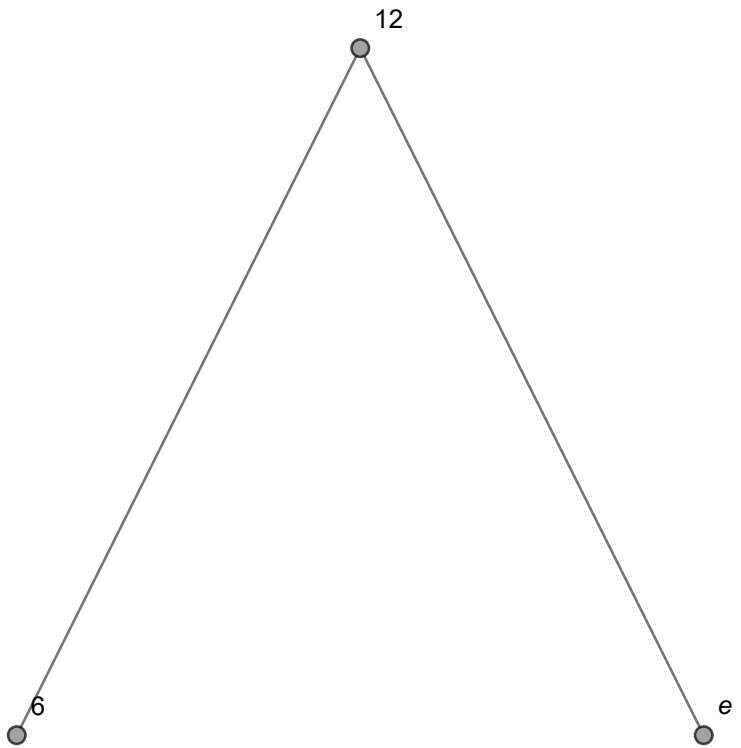
Se ha conservado un 3 como nodo del árbol anterior, pues $f_1 = 3$ no está relacionada con la frecuencia del caracter "i", sino más bien, con el resultado de la suma de dos frecuencias mínimas.
Ahora, se toman las dos frecuencias menores de L, $f_2 = 4$ y $f_3 = 5$, donde $nf = 4 + 5 = 9$. L se actualiza así $L = \{6, 9, 6\}$ y siendo $f_2 = 4$ la frecuencia de "r" y $f_3 = 5$ la de "n", se forma el árbol:



(8)

Solución del ejemplo 6.3

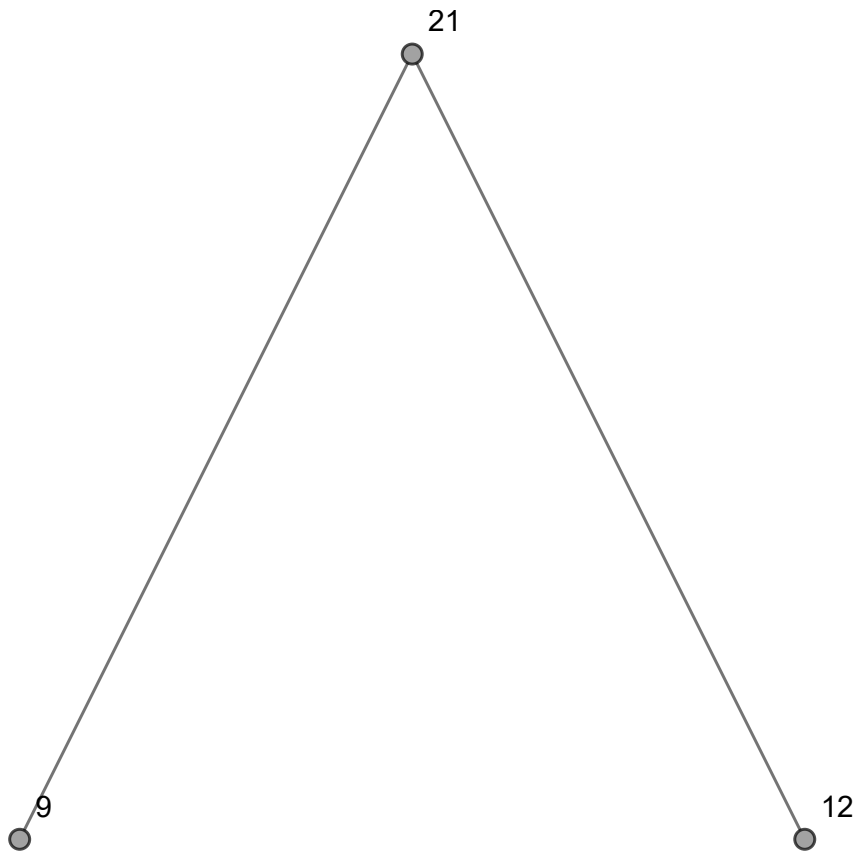
Luego, al elegir las dos frecuencias mínimas de L, $f_1 = 6$ y $f_3 = 6$, se tiene $nf = 6 + 6 = 12$, por lo que, $L = \{9, 12\}$ y donde $f_3 = 6$ se asocia al caracter "e", obteniéndose el árbol:



(9)

Solución del ejemplo 6.3

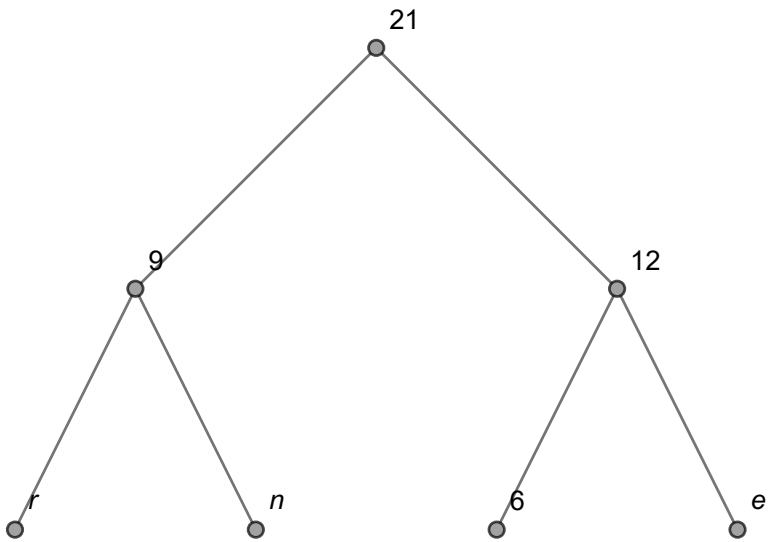
En la siguiente iteración, $nf = 9 + 12 = 21$, quedando el árbol:



(10)

Solución del ejemplo 6.3

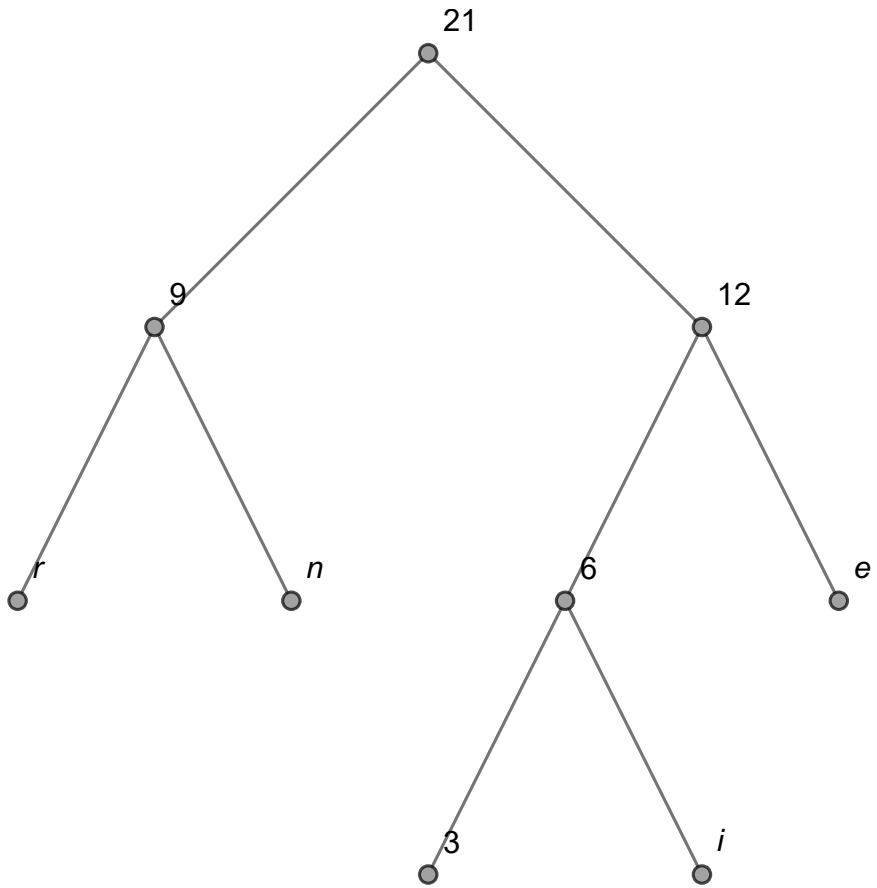
La lista L se actualiza como $L = \{21\}$ y al tener longitud igual a 1, se ha finalizado. El teorema 6 establece que un árbol de códigos de *Huffman* optimizado se estructura uniendo los distintos árboles encontrados en las iteraciones, yendo hacia atrás, es decir, se toma el árbol 10 y se adhieren los árboles 8 y 9, al poseer como raíces los vértices 9 y 12, respectivamente:



(11)

Solución del ejemplo 6.3

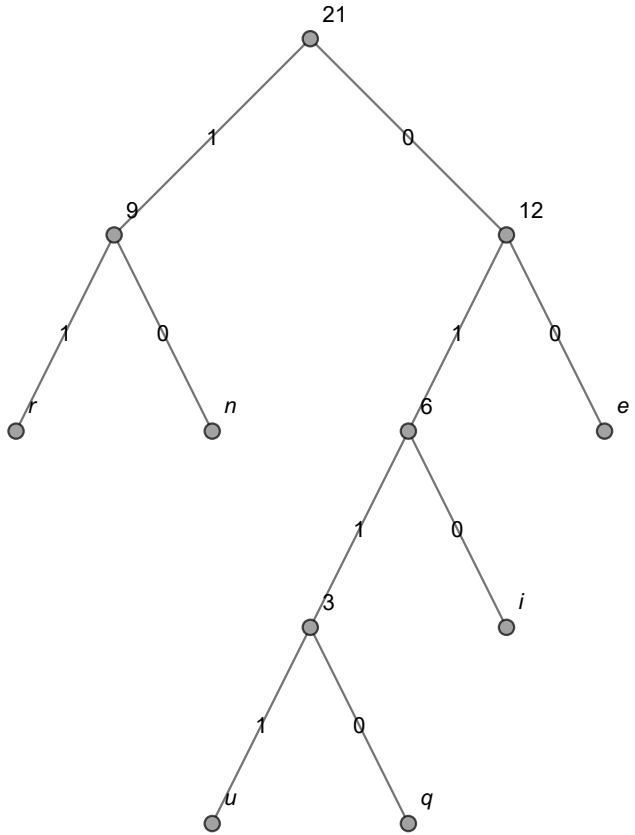
Al nodo 6 del árbol 11, se le une el árbol 7:



(12)

Solución del ejemplo 6.3

Luego, se adhiere a 12 el árbol 6 a través del nodo 3. Finalmente, un árbol de códigos de *Huffman* optimizado para la palabra "enrique" corresponde a:



(13)

Nota

Si bien es cierto, el árbol 13 no es equilibrado en un sentido estricto según la definición 1, este grafo es más apropiado al ejecutar sobre él operaciones con estructuras de datos, si le compara con el árbol 4, de allí que se considere a 13 un árbol optimizado.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-139.zip

Example (6.4)

Construya un árbol de códigos de *Huffman* no optimizado y optimizado para la palabra "informatica".

Solución del ejemplo 6.4

En este ejemplo el abecedario formado por la palabra "informatica" es $A = \{a, c, f, i, m, n, o, r, t\}$. Cabe aclarar que si la primera "a" de "informatica" hubiese estado tildada, se interpretaría como un caracter distinto. La tabla de frecuencias se expone a continuación:

Caracter	**Frecuencia**
i	2
n	1
f	1
o	1
r	1
m	1
a	2
t	1
c	1

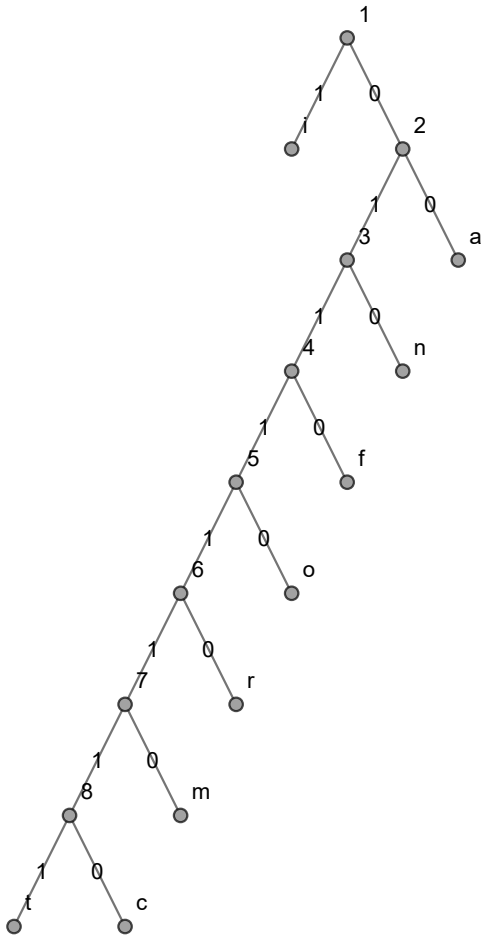
(14)

Solución del ejemplo 6.4

Un árbol de códigos de *Huffman* no optimizado se construye colocando en el nivel 1 la letra "i" o "a", donde si se elige el caracter "i" en el siguiente nivel se coloca la letra "a", o bien, si se pone primero la "a" en el siguiente nivel se añade la letra "i". Los demás caracteres al tener todos frecuencia 1, dan la libertad de integrarlos al árbol en cualquier orden a partir del nivel 3.

Solución del ejemplo 6.4

Existen varios árboles de códigos de *Huffman* no optimizados, uno de ellos es:



(15)

Solución del ejemplo 6.4

Si se desea generar ahora un árbol de códigos de *Huffman* optimizado, hay que reasignar las frecuencias de los caracteres mostrados en la tabla 14, con el objetivo de evitar repeticiones en dichos valores. Se aprecia que las mayores frecuencias deben estar vinculadas a las letras "a" e "i" que son las que poseen mayor cantidad de apariciones en la palabra "informatica".

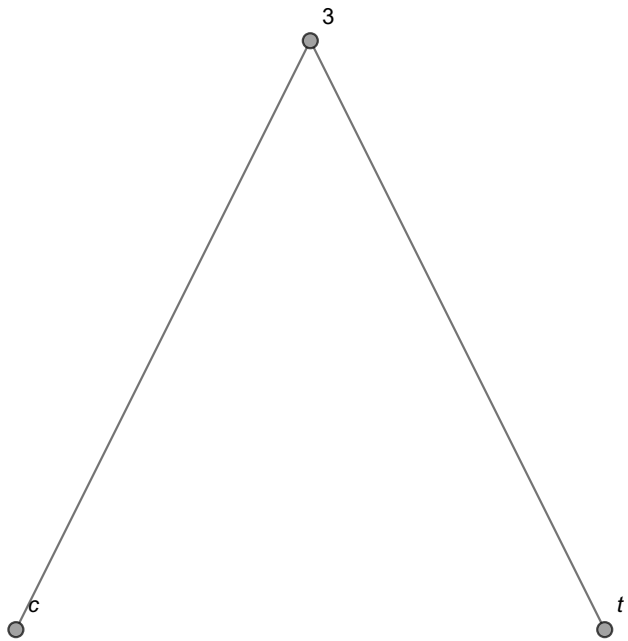
Solución del ejemplo 6.4

Existen varias posibilidades de reasignación, una de ellas corresponde a:

Caracter	Frecuencia reasignada
c	1
t	2
m	3
r	4
o	5
f	6
n	7
a	8
i	9

Solución del ejemplo 6.4

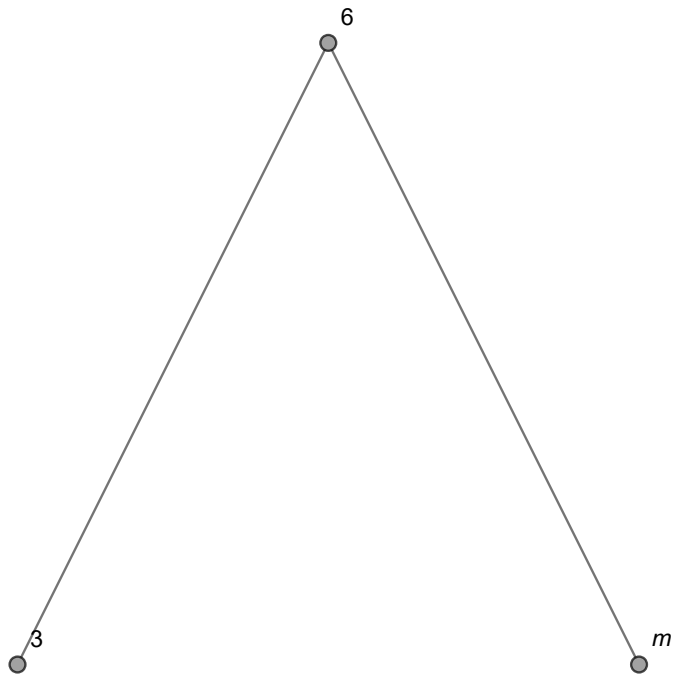
La lista inicial de frecuencias L quedaría de la forma $L = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Sea f_i la frecuencia de L ubicada en la posición i. Las dos frecuencias más pequeñas de L son $f_1 = 1$ y $f_2 = 2$, con lo cual, $nf = 1 + 2 = 3$. La frecuencia $f_1 = 1$ está relacionada con la letra "c" y la frecuencia $f_2 = 2$ está vinculada con el caracter "t" obteniéndose el árbol binario:



(16)

Solución del ejemplo 6.4

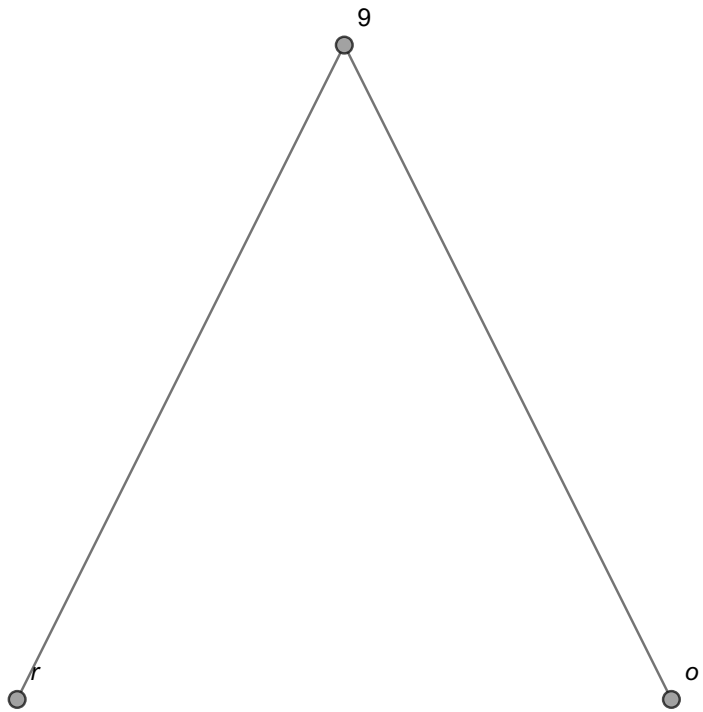
La lista L se actualiza como $L = \{3, 3, 4, 5, 6, 7, 8, 9\}$, en ella, las frecuencias mínimas son $f_1 = 3$ y $f_2 = 3$, en cuyo caso, $nf = 3 + 3 = 6$, la lista L se reduce a $L = \{6, 4, 5, 6, 7, 8, 9\}$ y al ser $f_2 = 3$ la frecuencia de "m", se deriva el árbol:



(17)

Solución del ejemplo 6.4

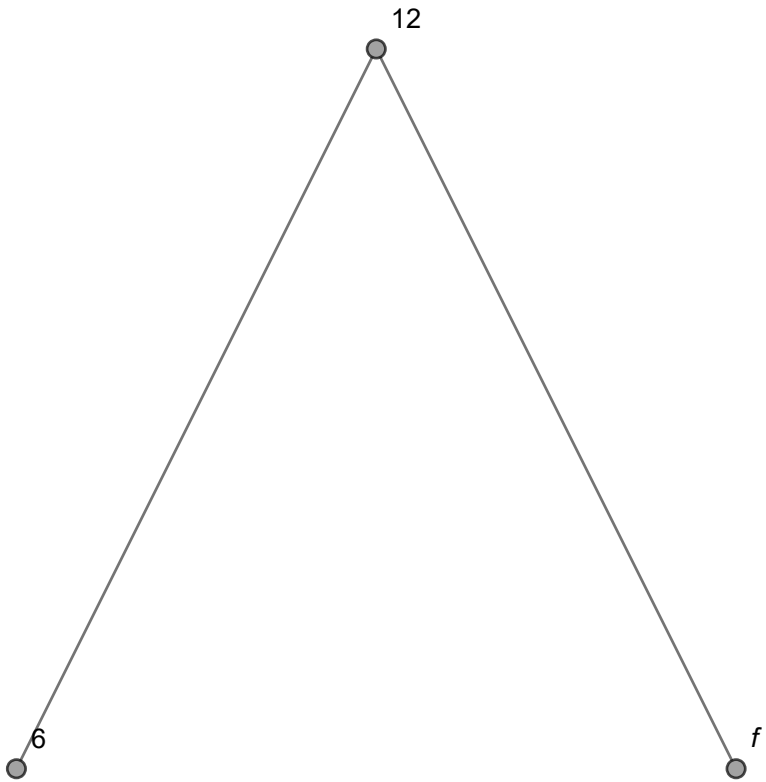
Ahora, las dos frecuencias más pequeñas de L son $f_2 = 4$ y $f_3 = 5$, por lo que, $nf = 4 + 5 = 9$ y $L = \{6, 9, 6, 7, 8, 9\}$. Además, $f_2 = 4$ es la frecuencia del caracter "r" y $f_3 = 5$ es la frecuencia de la letra "o", quedando el árbol binario:



(18)

Solución del ejemplo 6.4

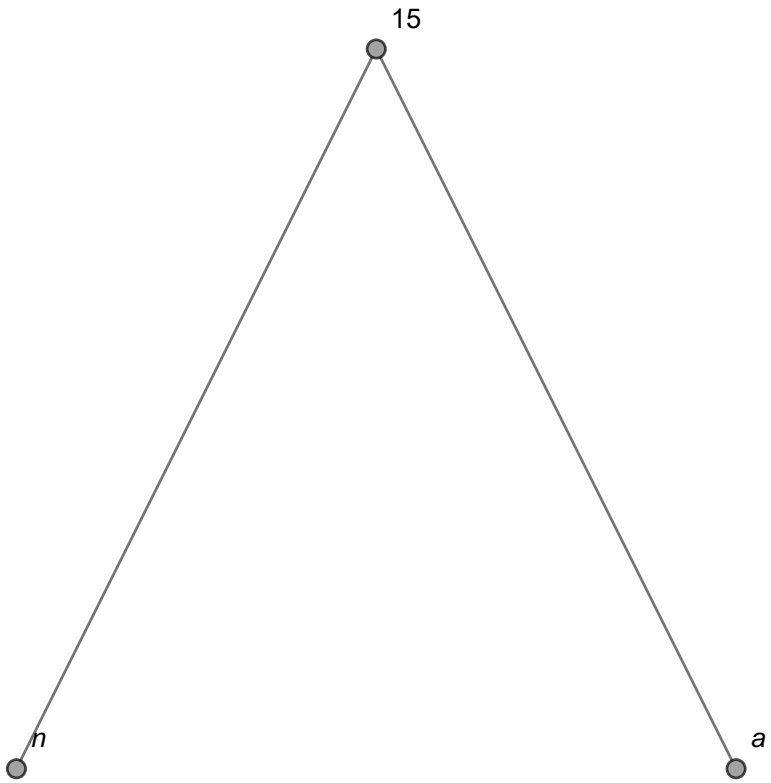
Luego, las dos frecuencias mínimas de L son $f_1 = 6$ y $f_3 = 6$, de donde, $nf = 6 + 6 = 12$, L se actualiza así $L = \{9, 12, 7, 8, 9\}$ y $f_3 = 6$ es la frecuencia de "f" formándose el árbol:



(19)

Solución del ejemplo 6.4

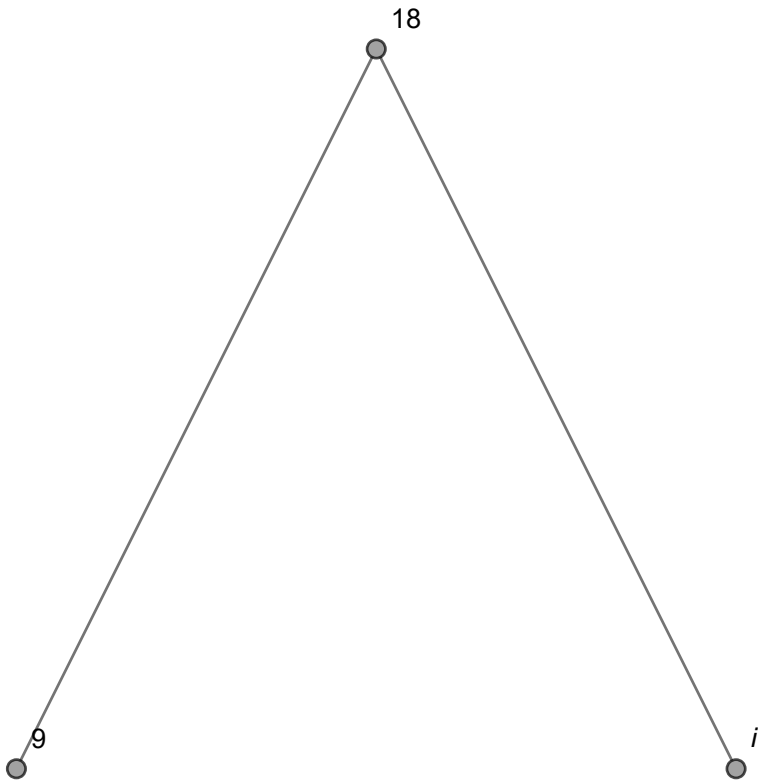
En L las dos frecuencias menores son $f_3 = 7$ y $f_4 = 8$, $nf = 7 + 8 = 15$ y $L = \{9, 12, 15, 9\}$. Además, $f_3 = 7$ se vincula a "n" y $f_4 = 8$ se asocia a "a", construyéndose el árbol:



(20)

Solución del ejemplo 6.4

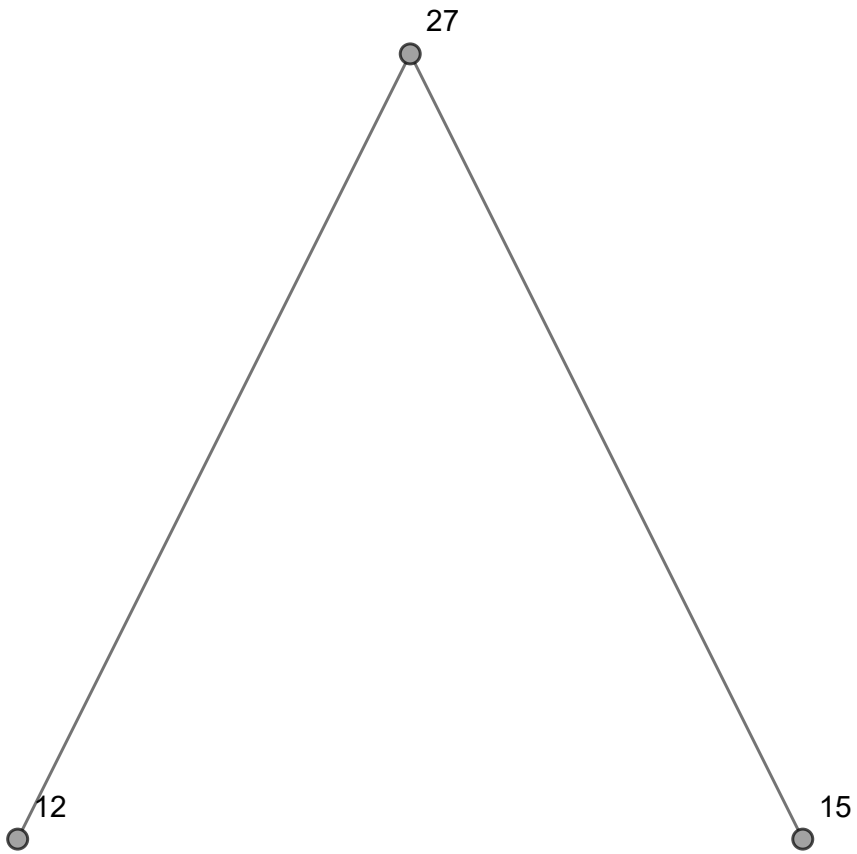
Ahora, las dos frecuencias más pequeñas de L son $f_1 = 9$ y $f_4 = 9$, f_4 relacionada con la letra "i", $nf = 9 + 9 = 18$ y $L = \{12, 15, 18\}$, quedando el árbol binario:



(21)

Solución del ejemplo 6.4

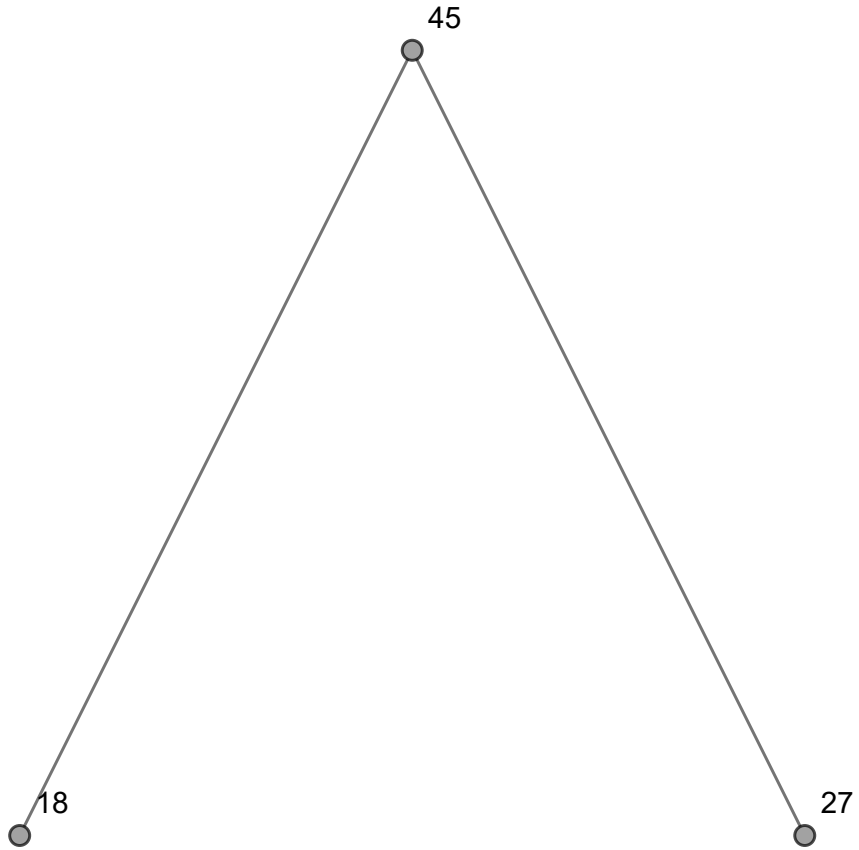
En L las dos frecuencias mínimas son $f_1 = 12$ y $f_2 = 15$, en consecuencia, $nf = 12 + 15 = 27$ y se forma el árbol:



(22)

Solución del ejemplo 6.4

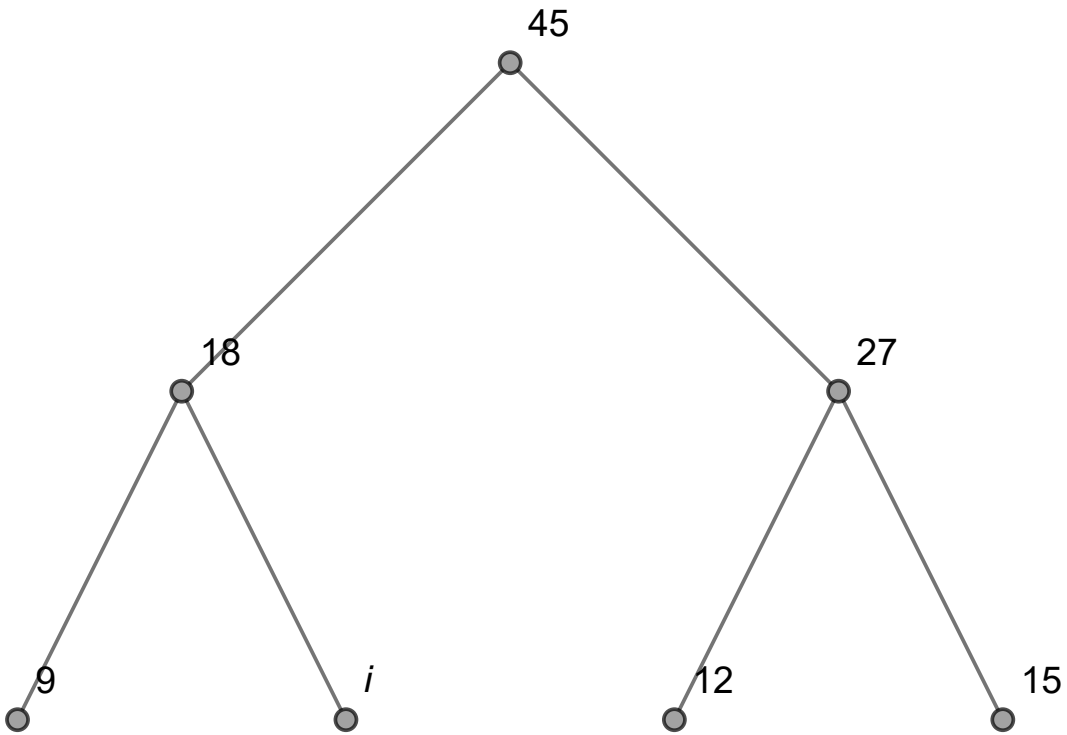
En esta nueva iteración, $L = \{18, 27\}$, $nf = 18 + 27 = 45$ y se genera el árbol:



(23)

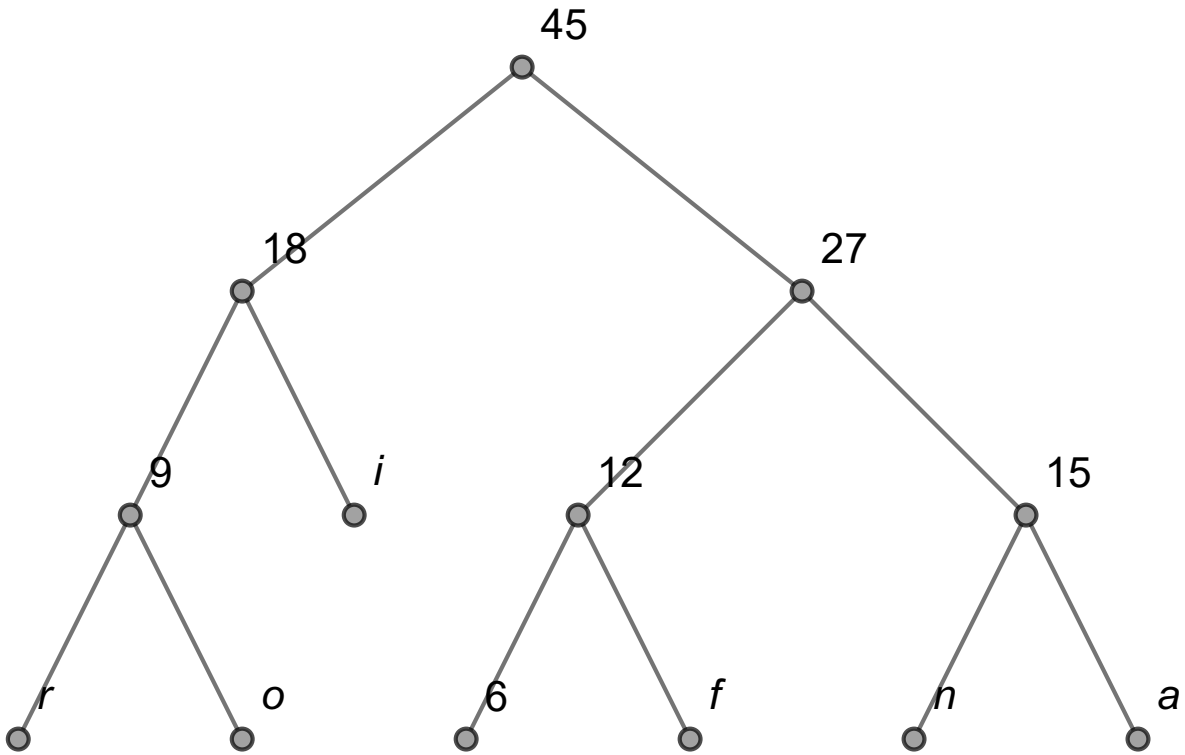
Solución del ejemplo 6.4

Se ha finalizado pues $L = \{45\}$ es de longitud 1. Para formar el árbol de códigos de *Huffman* optimizado se parte del grafo 23, adheriendo los árboles 21 y 22, por medio de los vértices 18 y 27, respectivamente. Luego:



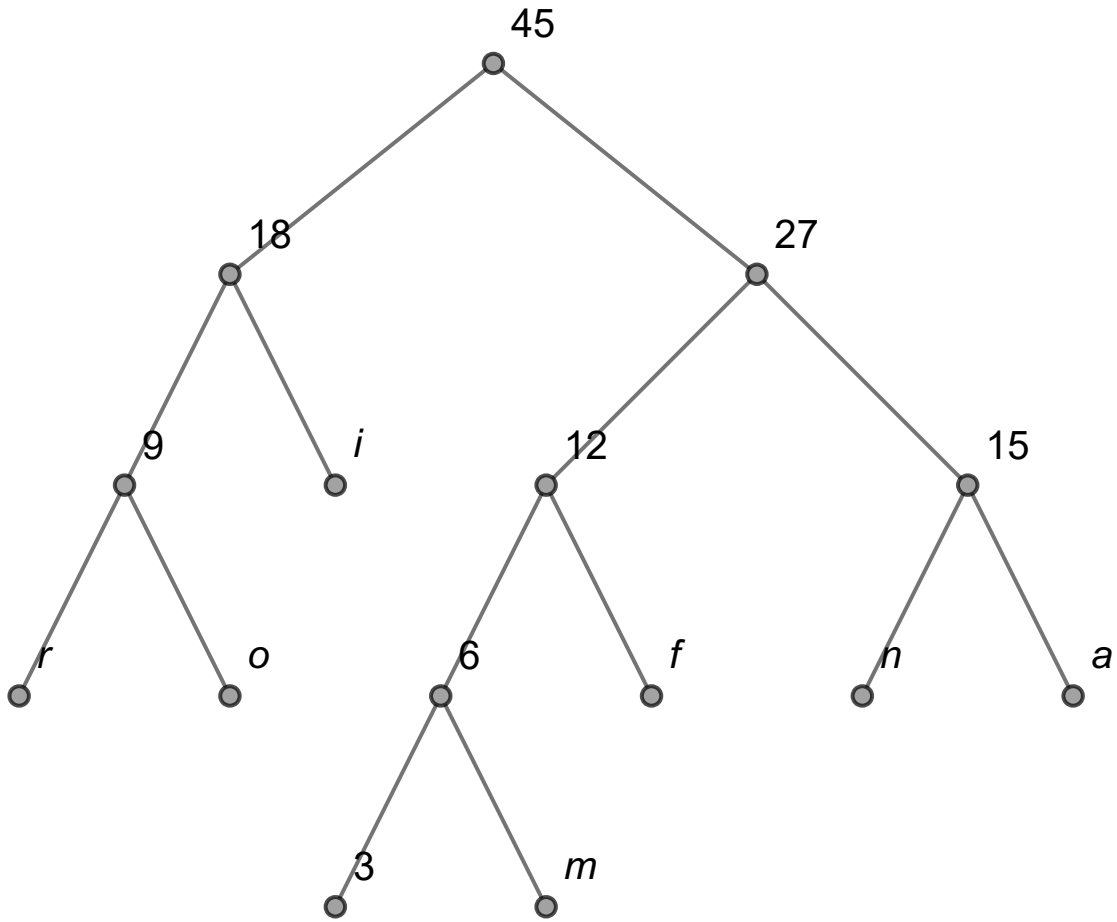
Solución del ejemplo 6.4

Se prosigue añadiendo los árboles 18, 19 y 20, por medio de los nodos 9, 12 y 15, respectivamente. Con lo cual se estructura el árbol:



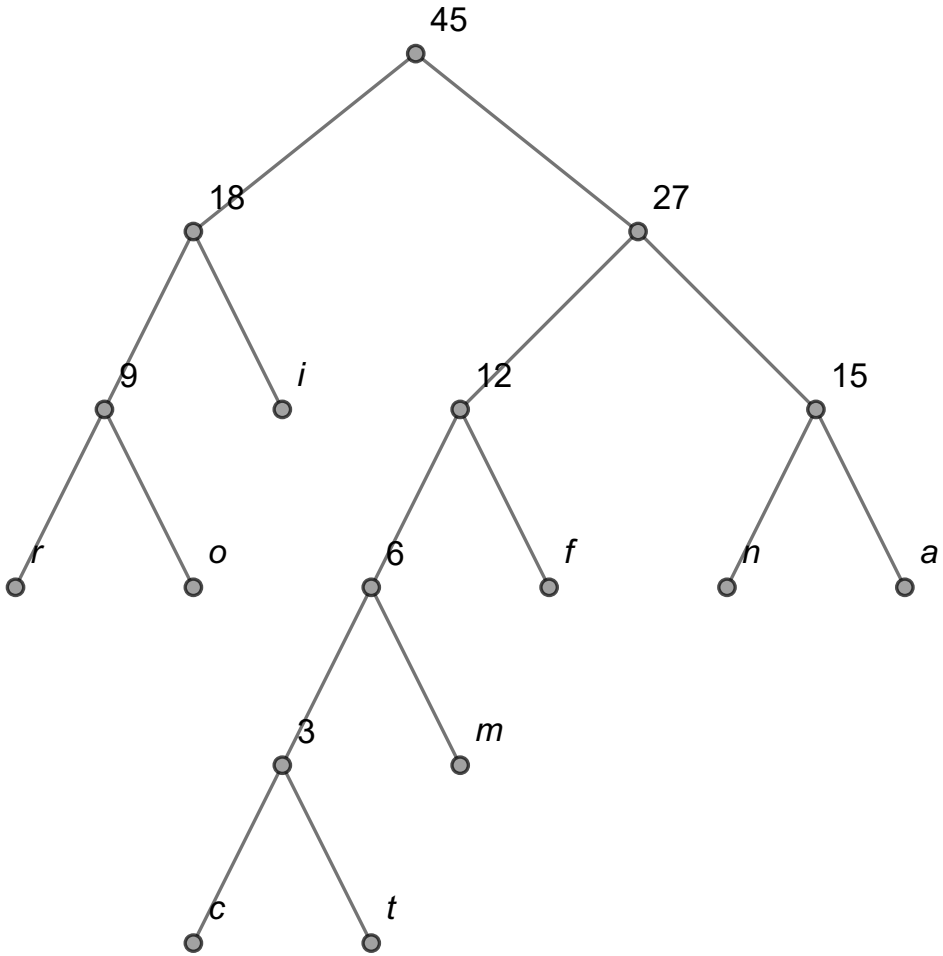
Solución del ejemplo 6.4

Ahora, se agrega el árbol 17 a través del vértice 6 :



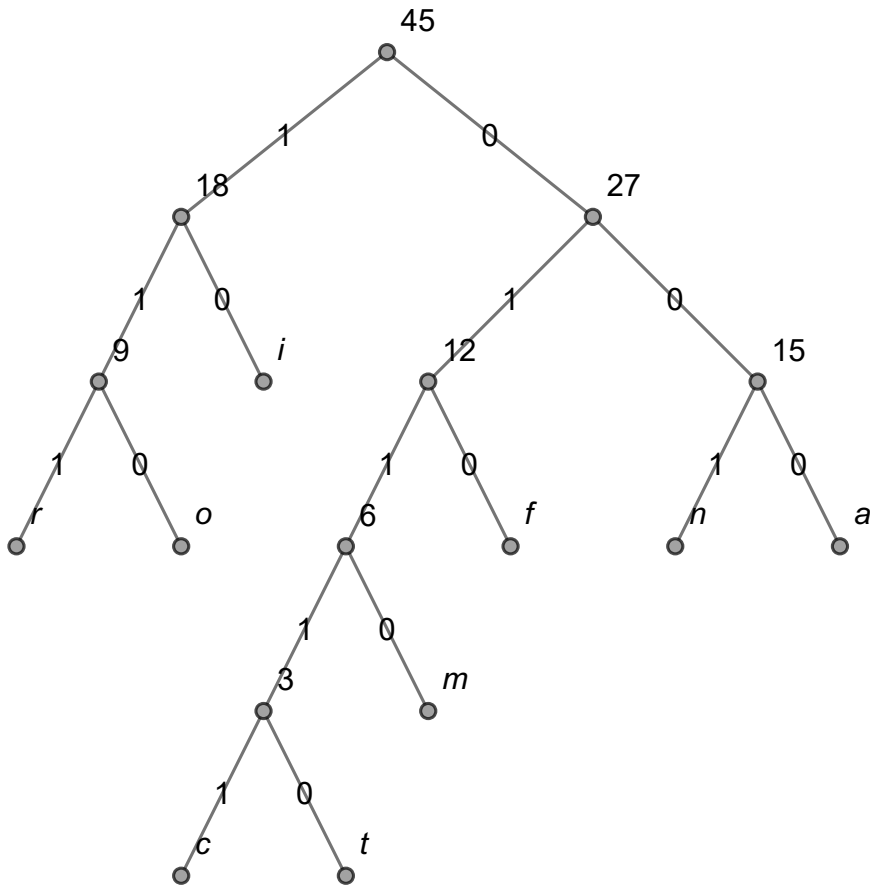
Solución del ejemplo 6.4

Y finalmente, se adhiere el árbol 16 usando el nodo 3 :



Solución del ejemplo 6.4

Luego, el árbol de códigos de Huffman optimizado obtenido, corresponde a:



(24)

Solución del ejemplo 6.4

El grafo 24 es optimizado a pesar de no cumplir con la definición de árbol balanceado (señalada en 1), dado que es más adecuado realizar sobre él operaciones con estructuras de datos, al compararlo con el árbol 15 no optimado.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-140.zip

Example (6.5)

Utilizando el software *Wolfram Mathematica* halle un árbol de códigos de *Huffman* no optimizado y optimizado para el *string* "felicidad".

Solución del ejemplo 6.5

El paquete **VilCretas** contiene dos interesantes comandos llamados: `ArbolHuffmanN` y `ArbolHuffman`. El primero determina sobre un *string* un árbol de códigos de *Huffman* no optimizado y el segundo, encuentra un árbol de códigos de *Huffman* optimizado al pasar como argumento una palabra. Ambas instrucciones cuentan con la opción `frecuencias -> True` que facilita visualizar las frecuencias asociadas a cada una de las letras del *string*. En este ejercicio:

In[ ] :=

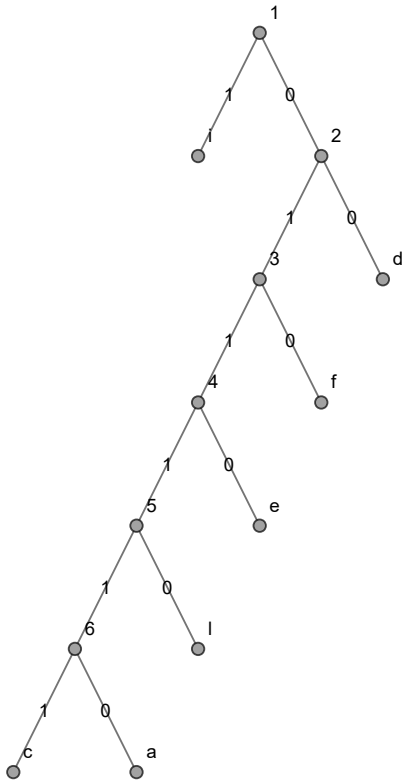
```
ArbolHuffmanN["felicidad", frecuencias -> True]
ArbolHuffman["felicidad", frecuencias -> True]
```

Solución del ejemplo 6.5

Se obtienen las siguientes salidas:

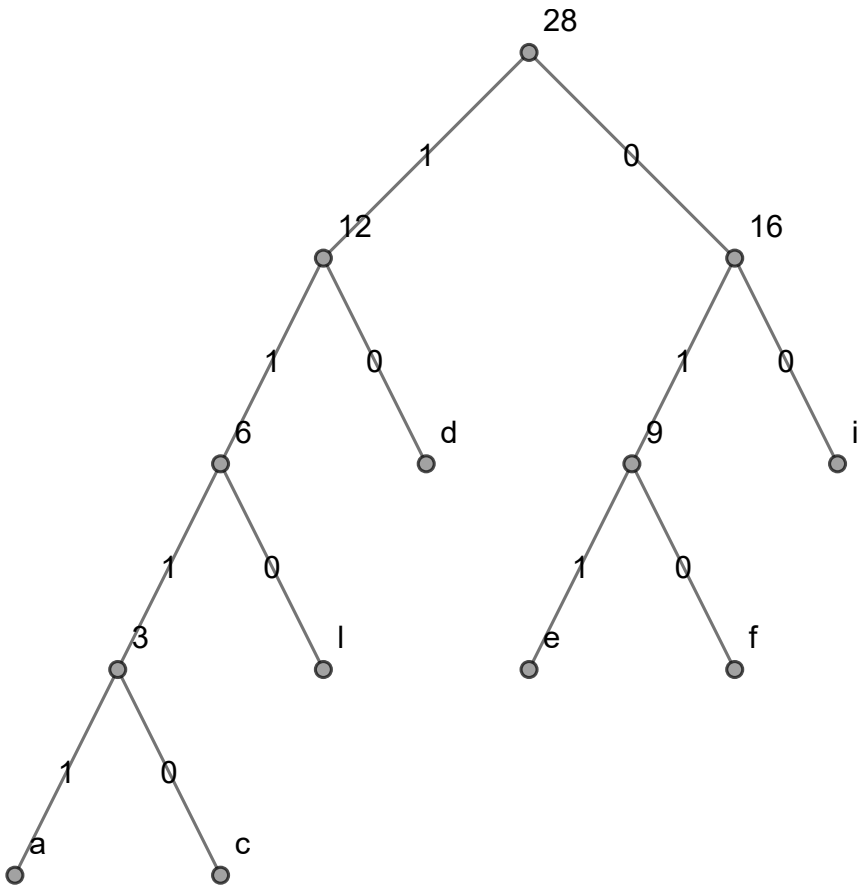
Out[ ] =

{{i,2},{d,2},{f,1},{e,1},{l,1},{c,1},{a,1}}



Solución del ejemplo 6.5

{{a,1},{c,2},{l,3},{e,4},{f,5},{d,6},{i,7}}



Solución del ejemplo 6.5

En el árbol no optimizado la instrucción `ArbolHuffmanN` emplea las frecuencias sin ninguna modificación, mientras que en el árbol optimizado `ArbolHuffman` realiza una reasignación de frecuencias.

Descargue un archivo

`https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-141.zip`

Uso del comando `ArbolHuffmanN`

Explicación en video

https://youtu.be/F5L5p2YGVIE

Empleo de la sentencia `ArbolHuffman`

Explicación en video

https://youtu.be/iMcnxuIvPGc

Los árboles binarios completos también se pueden utilizar en la representación de expresiones algebraicas. A estos árboles se les llama "árboles etiquetados". Para analizar esta aplicación de los árboles binarios se plantean dos ejemplos al respecto.

Example (6.6)

Determine un árbol etiquetado para la expresión algebraica:

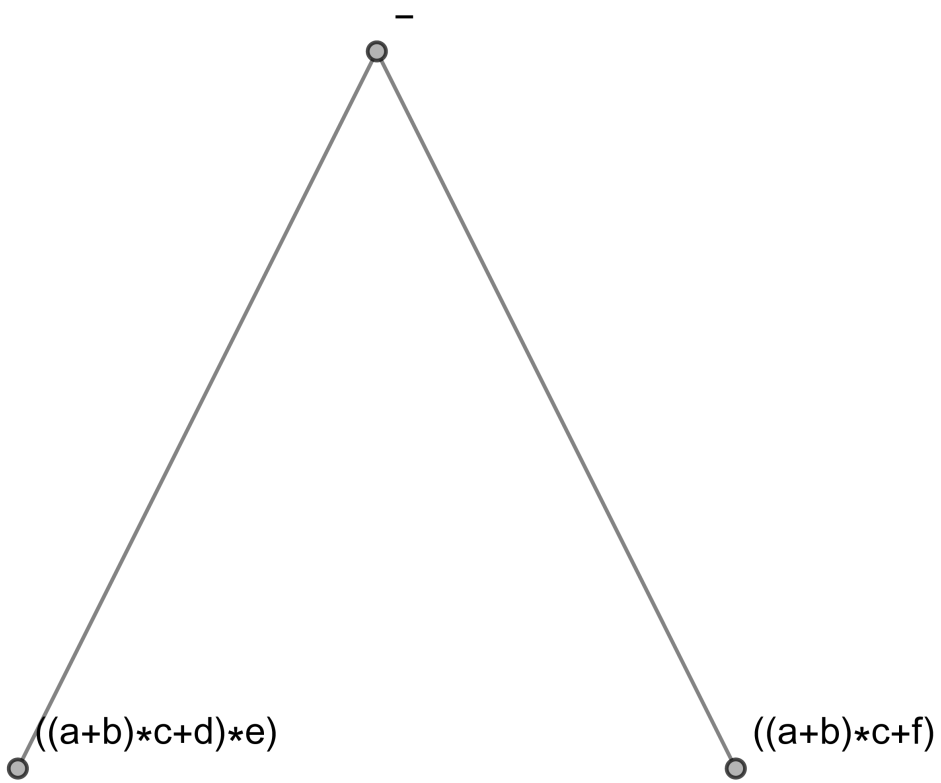
$$(((a+b)\cdot c+d)\cdot e)-((a+b)\cdot c+f)$$

Solución del ejemplo 6.6

El objetivo de un árbol etiquetado reside en representar las distintas operaciones realizadas dentro de una expresión algebraica. Para ello, normalmente la expresión debe ser leída de afuera hacia adentro con la intención de identificar las distintas ramas que constituirán el árbol. Una rama en este sentido con respecto a un vértice v, se entiende como el subgrafo obtenido al tomar a v, a uno o varios de sus nodos sucesores y a los lados que estos vértices determinan dentro del árbol.

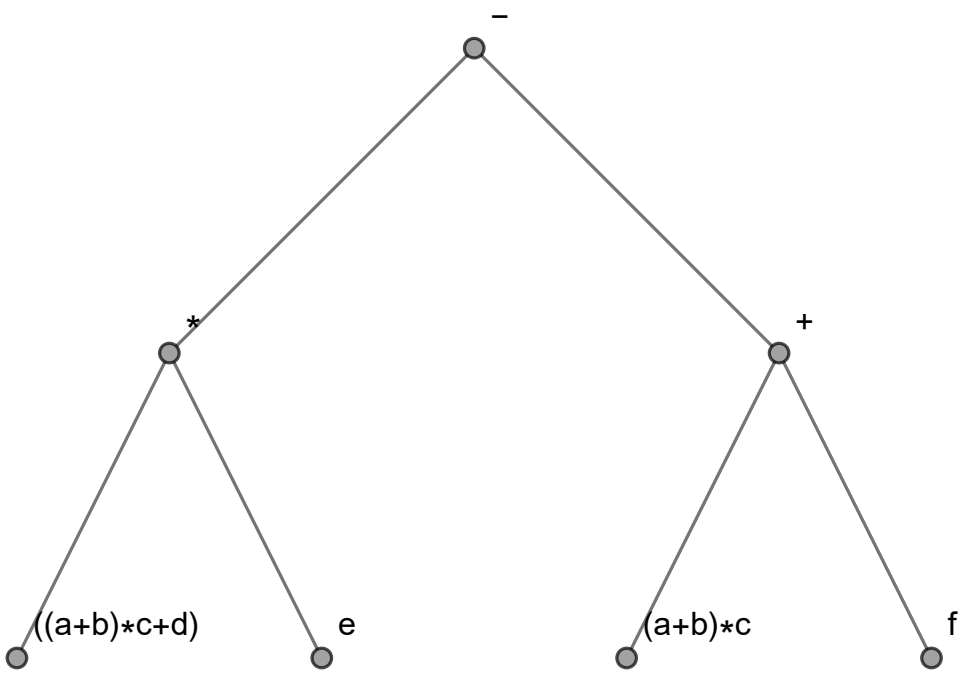
Solución del ejemplo 6.6

En este ejercicio, la operación de resta indica la existencia de dos ramas en el árbol a construir, una rama izquierda que podría contener la operación $((a+b)\cdot c+d)\cdot e$ y otra derecha, con $(a+b)\cdot c+f$, siendo el $-$ la raíz del árbol etiquetado:



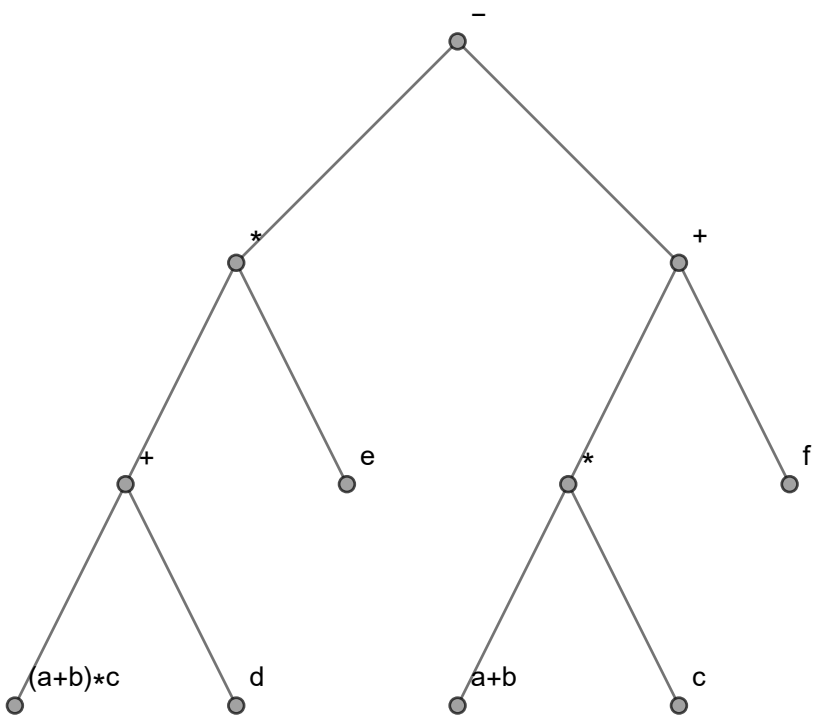
Solución del ejemplo 6.6

A su vez, en $((a+b)\cdot c+d)\cdot e$ se forman dos ramas, una para simbolizar $(a+b)\cdot c+d$ y la otra para e, con un nodo antecesor consecutivo igual a " $\cdot$ ". La expresión $(a+b)\cdot c+f$ se subdivide en dos ramas, una representa a $(a+b)\cdot c$ y la otra a f, con un vértice anterior correspondiente al operador $+$. Luego:



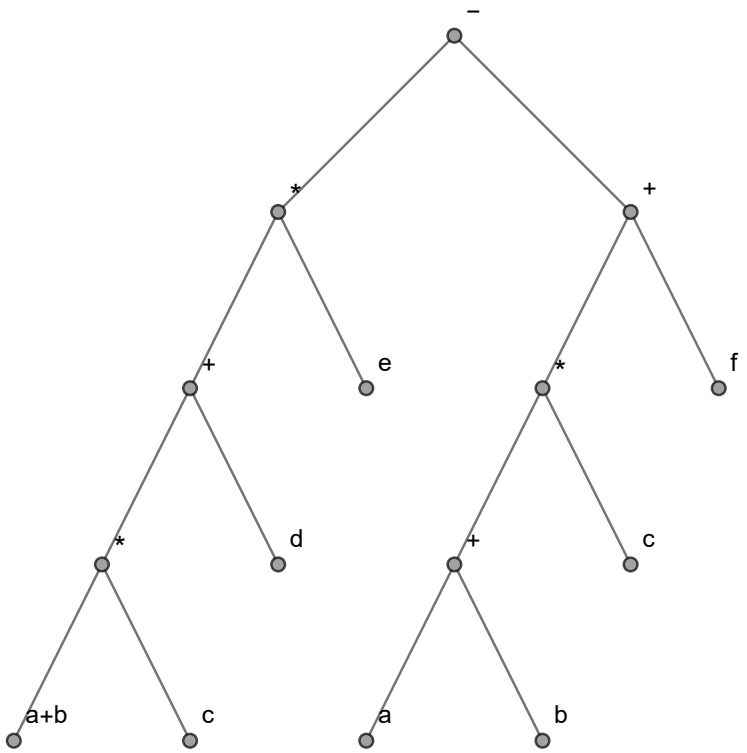
Solución del ejemplo 6.6

Del mismo modo, en $(a+b)\cdot c+d$ se derivan dos ramas, una que contiene $(a+b)\cdot c$ y la otra d, con un nodo antecesor consecutivo igual a $+$. Ahora, para simbolizar $(a+b)\cdot c$, se toman dos ramas, una contiene a $a+b$ y la otra el vértice c, con un nodo antecesor consecutivo igual a " $\cdot$ ". El árbol tomaría la forma:



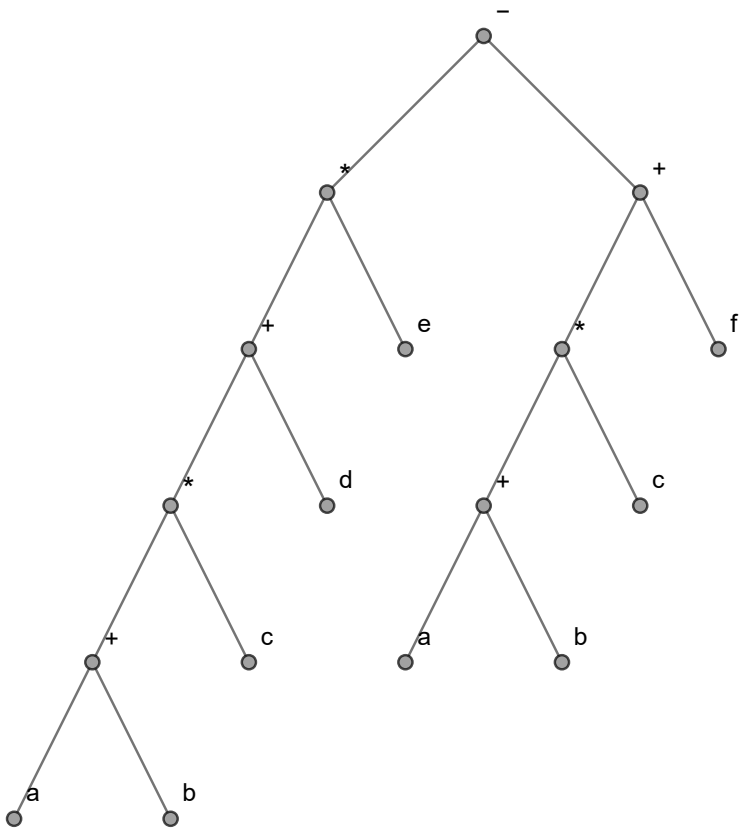
Solución del ejemplo 6.6

Ahora, de manera análoga, el $(a+b)\cdot c$ se descompone en dos ramas, una para $a+b$ y la otra para c, con un nodo anterior correspondiente a la operación " $\cdot$ ". También, $a+b$ se descompone en dos ramas cuyos vértices son a y b, con un $+$ como nodo padre. En consecuencia:



Solución del ejemplo 6.6

Luego, $a + b$ se divide en dos ramas, una con a y la otra con b, con un vértice antecesor consecutivo igual a $+$. Finalmente, se obtiene como resultado de este proceso el árbol etiquetado:



Solución del ejemplo 6.6

Este grafo en este ejemplo no es único, pues cuando se simboliza una operación conmutativa como la suma, o bien, la multiplicación, la rama izquierda podría ubicarse al lado derecho y la rama derecha al lado izquierdo, inviertiéndose el orden.

Nota

En un árbol etiquetado, los nodos internos son las operaciones de la expresión algebraica, mientras que los vértices externos u hojas, están etiquetados con los términos que se operan.

Solución del ejemplo 6.6

Wolfram Mathematica permite hallar un $k-$árbol etiquetado de una expresión algebraica mediante el comando `ArbolExpAlgebraica` de la librería **VilCretas**. En este ejercicio:

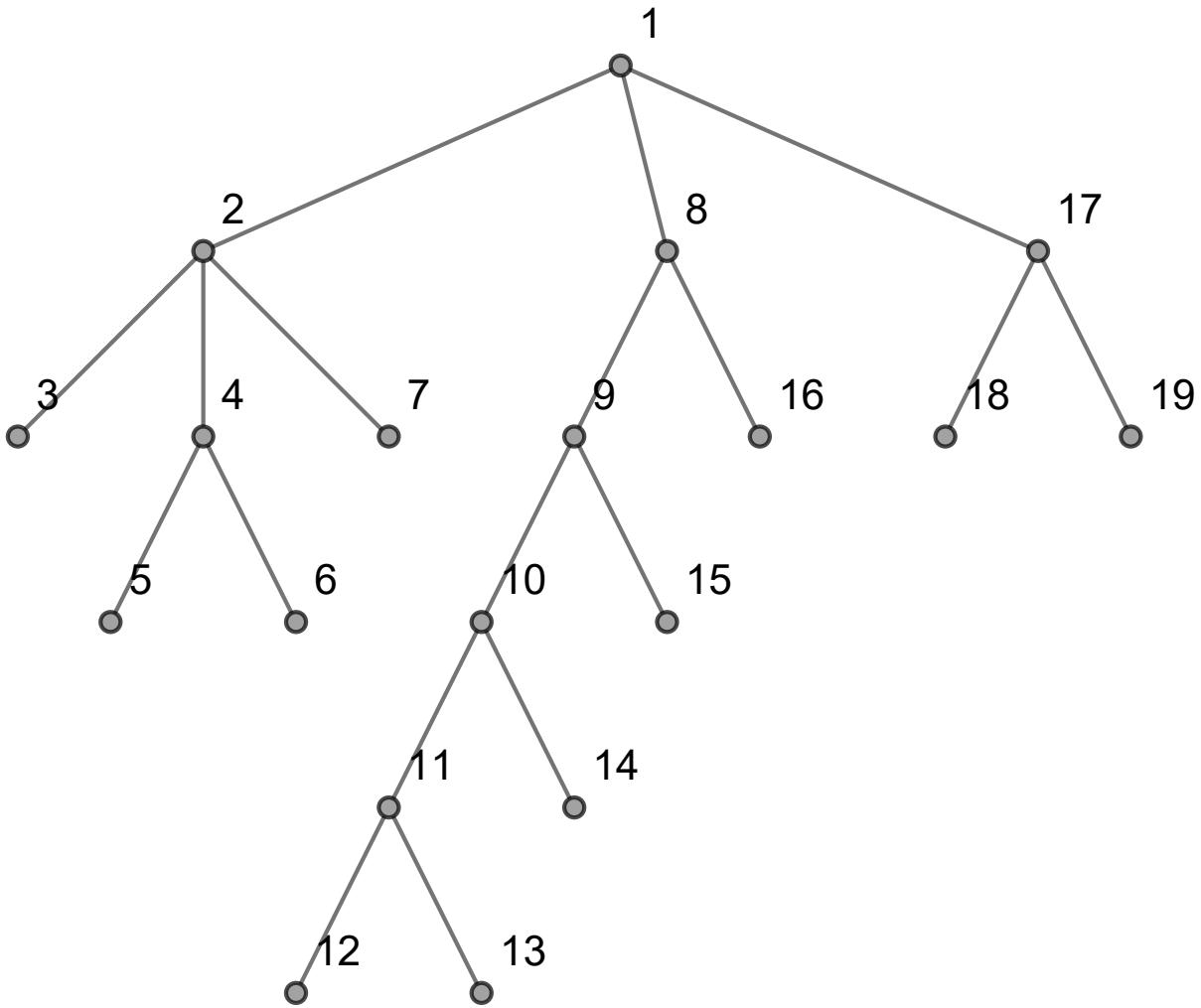
In[ ] :=

```
ArbolExpAlgebraica[(((a + b) c + d) e) - ((a + b) c + f)]
```

Out[ ] =

```
Plus[Times[-1, Plus[a, b], c], Times[Plus[Times[Plus[a, b], c], d], e],
Times[-1, f]]
{1->Plus, 2->Times, 3->-1, 4->Plus, 5->a, 6->b, 7->c, 8->Times,
9->Plus,
10->Times, 11->Plus, 12->a, 13->b, 14->c, 15->d, 16->e, 17->Times,
18->-1,
19->f}
```

Solución del ejemplo 6.6



Solución del ejemplo 6.6

La primera línea del **Out[ ]** corresponde a la expresión algebraica utilizada por el software en el diseño del árbol, donde Plus simboliza la operación suma y Times la operación multiplicación. Power en otros ejercicios, suplanta el operador de potenciación. De acuerdo con ello, en este ejemplo, *Mathematica* está usando en la construcción del árbol etiquetado la expresión algebraica:

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

La segunda línea del **Out[ ]** aclara el significado de los vértices del árbol etiquetado devuelto por `ArbolExpAlgebraica`. Esta sentencia, algunas veces, no retorna un árbol binario. De hecho, el árbol mostrado en la salida anterior es de orden 3.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-142.zip

Explicación en video

https://youtu.be/UvssVh9y47w

Example (6.7)

Halle una representación mediante un árbol binario completo para la expresión algebraica:

$$(3-2a)+(b\div 2c-3d+e)+\frac{(f-4)\,c}{g}+a$$

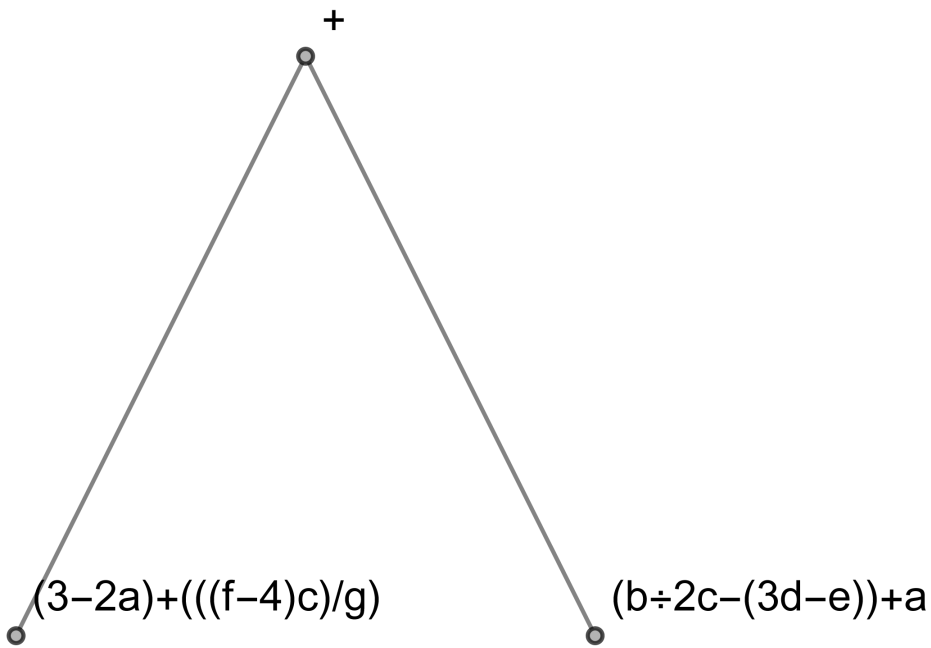
Solución del ejemplo 6.7

En este ejemplo, antes de iniciar el proceso de desarrollo del árbol etiquetado es necesario realizar una asociación en los sumandos que constituyen la expresión algebraica. Hay varias formas de emprender esto, por lo que existen varios árboles etiquetados como respuesta. Supongamos que se parte de la siguiente expresión algebraica:

$$\left[(3-2a)+\frac{(f-4)\,c}{g}\right]+[(b\div 2c-(3d-e))+a] \tag{25}$$

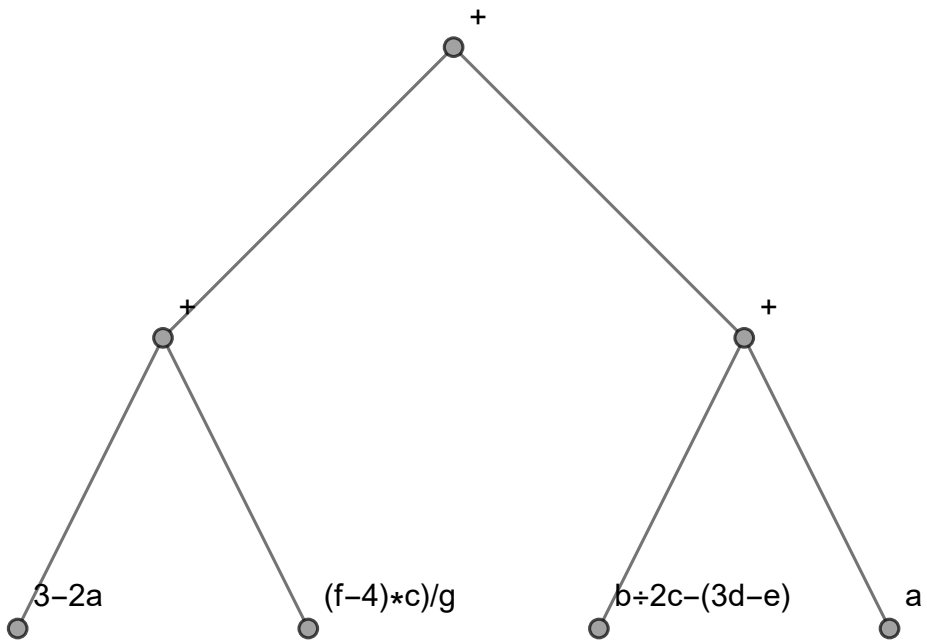
Solución del ejemplo 6.7

El $+$ fuera de los corchetes cuadrados es la raíz del árbol de interés. Al lado izquierdo del $+$ se representa a $(3-2a)+\frac{(f-4)c}{g}$ y al lado derecho a $(b \div 2c-(3d-e))+a$:



Solución del ejemplo 6.7

En $(3-2a)+\frac{(f-4)c}{g}$ se extrae el $+$ fuera de los paréntesis redondos, quedando una rama que simboliza a $3-2a$ y otra que representa a $\frac{(f-4)c}{g}$. Para $(b \div 2c-(3d-e))+a$ se coloca el $+$ fuera de los paréntesis redondos como vértice antecesor consecutivo, a un lado se toma una rama para simbolizar $b \div 2c-(3d-e)$ y al otro se coloca a. Por lo tanto:

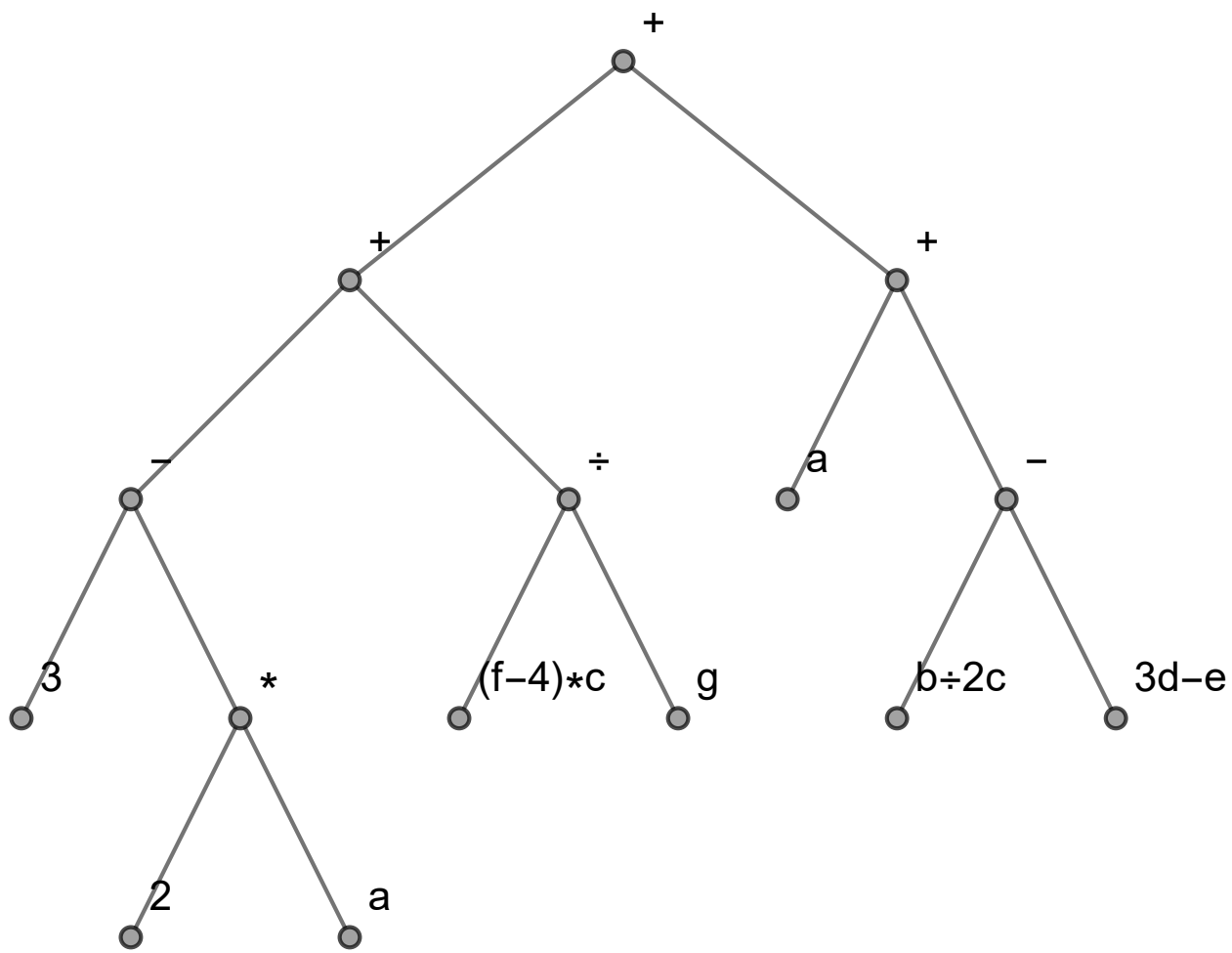


Solución del ejemplo 6.7

En $3-2a$, se sitúa el $-$ como el nodo antecesor consecutivo y se agregan dos ramas, una a la izquierda con extremo 3 y la otra a la derecha para simbolizar el $2a$, donde $2a$ se descompone tomando un " $\cdot$ " en el vértice antecesor consecutivo, un 2 a la izquierda y una a a la derecha. Por otra parte, $\frac{(f-4)c}{g}$ se simboliza con dos ramas, al lado izquierdo para representar $(f-4)\,c$ y al lado derecho g, con un nodo anterior igual a $\div$. Luego, $b \div 2c-(3d-e)$ se descompone mediante dos ramas, al lado izquierdo se toma $b \div 2c$ y al lado derecho $3d-e$, poniendo $-$ como el nodo anterior.

Solución del ejemplo 6.7

En consecuencia, se obtiene el árbol:

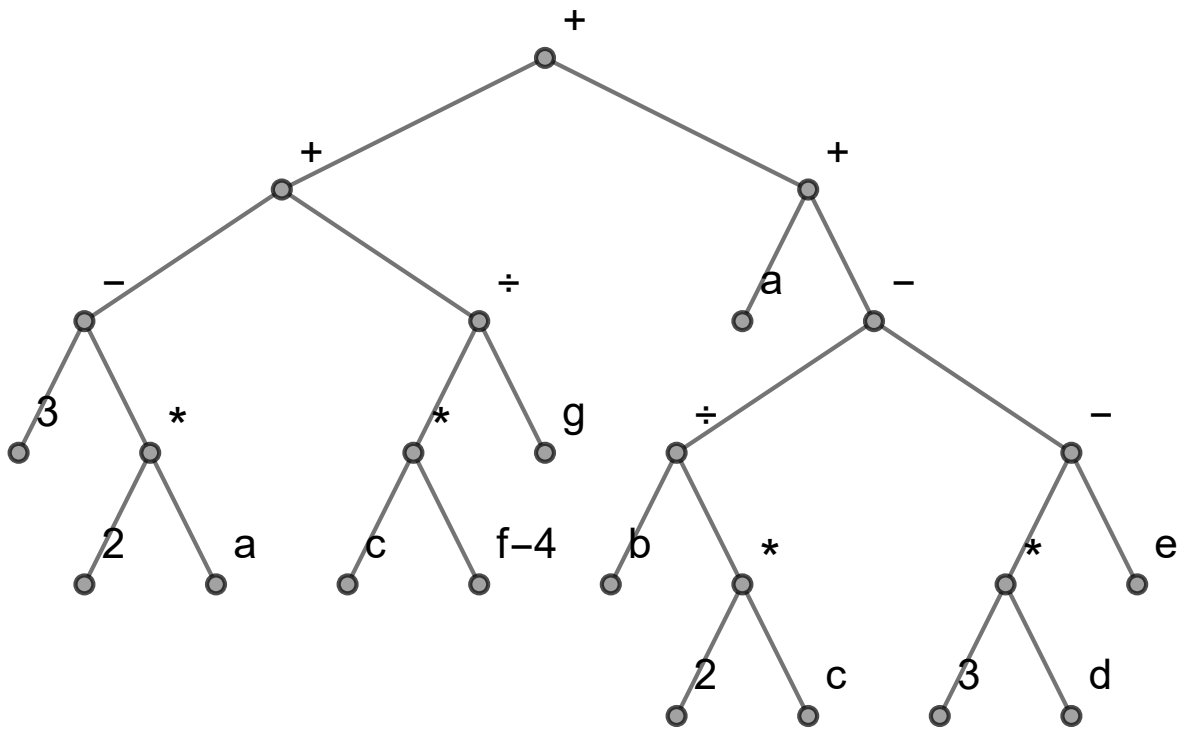


Solución del ejemplo 6.7

Ahora, $(f-4)\,c$ requiere el uso de dos ramas, a un lado se pone el $f-4$ y al otro el nodo c, con un vértice antecesor consecutivo igual a " $\cdot$ ". La expresión $b \div 2c$ implica el uso de dos ramas, al lado izquierdo queda b y al lado derecho un " $\cdot$ " como nodo padre de 2 y c, con $\div$ un vértice antecesor consecutivo. Además, de forma análoga, $3d - e$ deriva dos ramas, al lado izquierdo se coloca un " $\cdot$ " como padre de 3 y d y, al lado derecho queda el nodo e, con $-$ un vértice antecesor consecutivo.

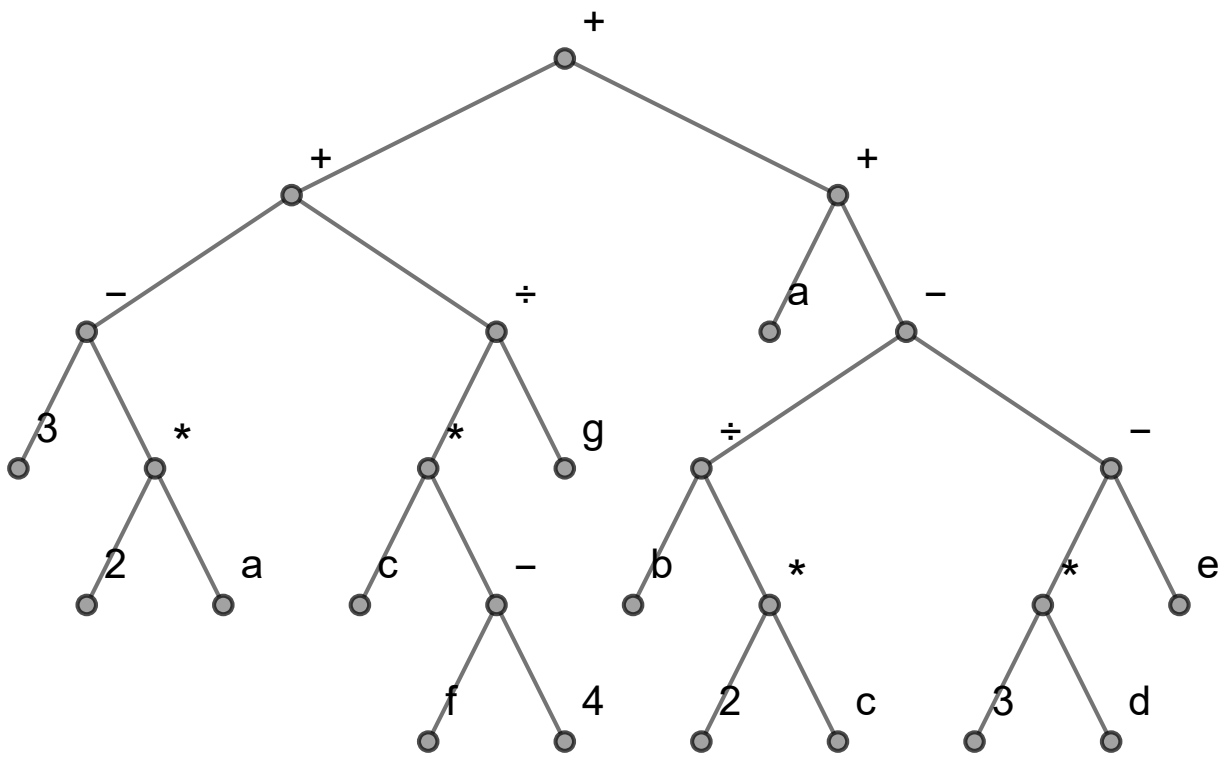
Solución del ejemplo 6.7

Originándose el árbol:



Solución del ejemplo 6.7

Finalmente, $f - 4$ implica el uso de dos ramas una con extremo f al lado izquierdo y la otra, con extremo 4 al lado derecho, siendo el nodo anterior igual a $-$. Un árbol etiquetado para la expresión algebraica 25, corresponde a:



Solución del ejemplo 6.7

En este ejercicio, la sentencia `ArbolExpAlgebraica` ofrece como salida un árbol etiquetado de orden 6 :

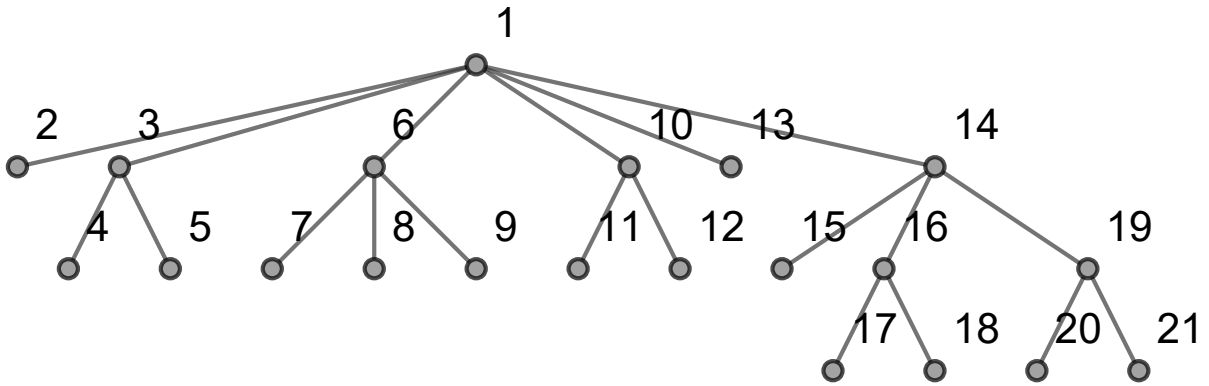
In[ ] :=

```
ArbolExpAlgebraica[((3 - 2 a) + (((f - 4) c)/g)) + ((b/2 c
- (3 d - e)) + a)]
```

Out[ ] =

Plus[3, Times[-1, a], Times[Rational[1, 2], b, c], Times[-3, d], e,
Times[c, Plus[-4, f], Power[g, -1]]]
{1->Plus, 2->3, 3->Times, 4->-1, 5->a, 6->Times, 7->1/2, 8->b,
9->c, 10->Times, 11->-3, 12->d, 13->e, 14->Times, 15->c, 16->Plus,
17->-4, 18->f, 19->Power,
20->g, 21->-1}

Solución del ejemplo 6.7



Se aprecia que la expresión algebraica empleada por el software para obtener el árbol etiquetado del **Out[ ]** es:

$$3 - a + \frac{b \cdot c}{2} - 3d + e + \frac{c \cdot (-4 + f)}{g}$$

Solución del ejemplo 6.7

Además, cabe señalar que la instrucción `ArbolExpAlgebraica` no admite el uso de paréntesis cuadrados ([]) o llaves ({}) al anidar expresiones, solo adopta el empleo de paréntesis redondos con esa finalidad.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-143.zip

- Los árboles etiquetados tienen un uso esencial al permitir encontrar distintas notaciones para una expresión algebraica, llamadas: “notación polaca” y “notación polaca inversa”. Este tema se relaciona con las formas de representación computacional que posee una expresión algebraica y se estudiará en una de las secciones posteriores.
- Otra importante aplicación de los árboles binarios, será desarrollada en la sección que prosigue y se sustenta en disponer este tipo de grafos como una “estructura de datos”.

Árboles binarios de búsqueda

Prof. Enrique Vílchez Quesada

Universidad Nacional de Costa Rica

Árboles binarios de búsqueda

Los árboles binarios suelen ser utilizados para almacenar información en sus vértices y ordenar dichos datos por medio de la estructura que adquiere el árbol. Esto es lo que se conoce como una "estructura de datos", o bien, en este contexto, un "árbol binario de búsqueda". Existen diversas operaciones que se pueden realizar sobre un árbol binario de búsqueda, tales como: inserción, búsqueda y eliminación. En este texto se contemplan únicamente las operaciones de inserción y búsqueda.

Algoritmo de creación de un árbol binario de búsqueda

Crear un árbol binario de búsqueda consiste en tomar una lista de datos distintos $Datos = \{v_1, v_2, v_3, \ldots, v_n\}$ y construir un árbol binario (no necesariamente completo) como se describe a continuación:

1. v_1 es la raíz.
2. Para saber si v_2 es un hijo izquierdo o derecho de v_1, se comparan ambos datos. Si $v_1 > v_2$, v_2 se coloca como un nodo a la izquierda de la raíz y de lo contrario, como un vértice a la derecha. De manera equivalente, el nodo v_3 de *Datos* se inserta comparando v_1 y v_3. Si $v_1 > v_3$, v_3 ocupa la dirección izquierda en el árbol. Suponiendo que a la izquierda se encuentra v_2, se pregunta si $v_2 > v_3$, en dicho caso, v_3 toma el lugar de un vértice a la izquierda de v_2, o bien, si $v_2 < v_3$, v_3 se coloca a la derecha de v_2. Si desde un inicio, $v_1 < v_3$, v_3 es un vértice derecho con respecto a v_1.
3. Se continua así, hasta insertar los n datos de *Datos*.

Nota

El algoritmo es aplicable si todos los datos de la lista *Datos* son distintos, si hay elementos repetidos el procedimiento descrito no correrá correctamente. Otra limitación que presenta, radica en que muchas veces no proporciona como salida un árbol equilibrado, lo cual sería lo idóneo con el objetivo de manipular la estructura de datos de una forma más eficiente bajo una perspectiva computacional. Posiblemente en cursos más avanzados el alumno estudiará métodos de balanceo que resuelven este tipo de problemas.

Consideremos algunos ejemplos.

Example (6.8)

Elabore un árbol binario de búsqueda para:

$$Datos = \{50, 7, 44, 83, 84, 2, 11, 95, 36, 60, 94, 29, 37, 100, 82, 55, 9, 32,$$

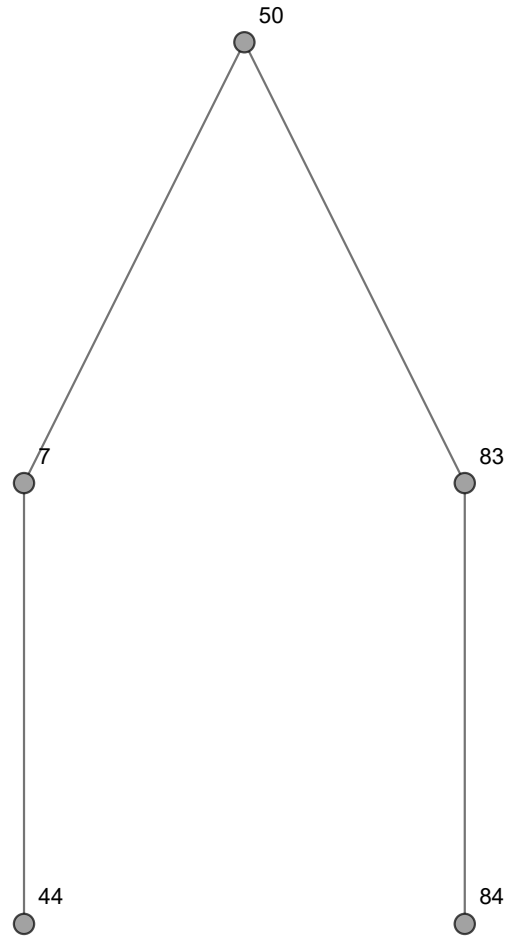
$$93, 74, 49, 98, 66\}$$

Solución del ejemplo 6.8

Se comienza tomando a 50 como la raíz del árbol binario de búsqueda. Luego, se inserta el dato 7, 50 es mayor que 7, por lo que 7 se coloca a la izquierda de 50. Se inserta el 44 comparando 50 con 44, como 44 es menor que 50 se sitúa a la izquierda de 50, donde al encontrarse el dato 7 se comparan y al ser 7 menor que 44, 44 se emplaza a la derecha de 7. Ahora, al insertar 83, 83 es mayor que 50 y por lo tanto, se coloca a su derecha. Al ingresar el dato 84, 50 es menor que 84 y 83 es menor que 84, ocasionando ubicar el 84 a la derecha de 83.

Solución del ejemplo 6.8

En este punto, el árbol adquiere la forma:



Solución del ejemplo 6.8

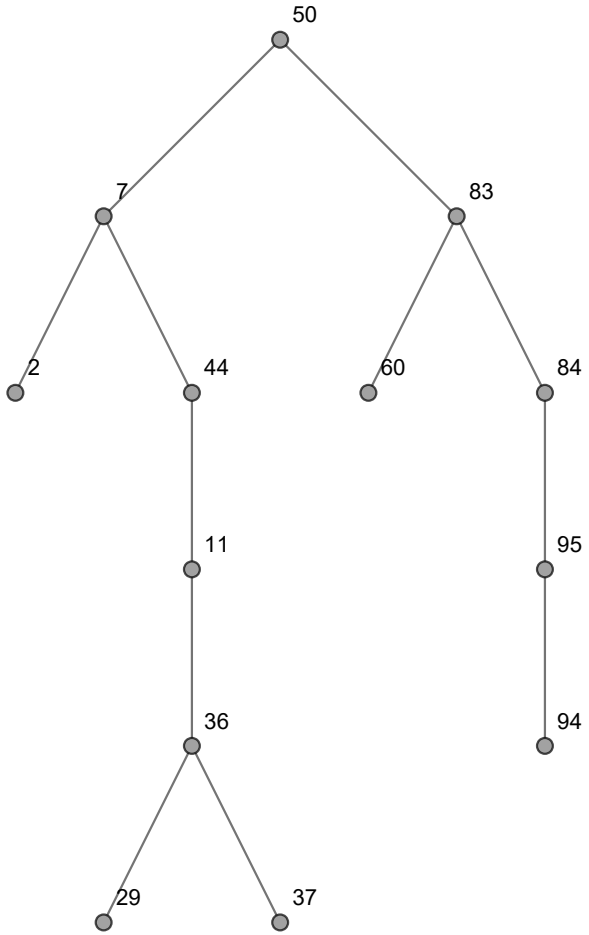
El árbol anterior fue construido con ayuda del software *Wolfram Mathematica*. En este programa si se tiene un nodo padre con un solo hijo, por defecto se dibuja la arista que los une de manera vertical, es decir, sin asumir ninguna dirección a la derecha o a la izquierda. Pese a ello, el alumno debe apreciar que las aristas $7 \bullet\!\!-\!\!\bullet 44$ y $83 \bullet\!\!-\!\!\bullet 84$ tienen una dirección a la derecha.

Solución del ejemplo 6.8

Continuando con el proceso, corresponde insertar el dato 2, 2 es menor que todos los datos ingresados hasta el momento, por lo que se coloca a la izquierda de 7. Al tomar a 11, 11 es menor que 50 se ubica a su izquierda, 11 es mayor que 7 se emplaza a su derecha y 11 es menor que 44 colocándose a la izquierda de 44. En el dato 95, 95 es mayor que 50, 83 y 84 ubicándose a la derecha de 84. El dato 36 es menor que 50, es mayor que 7, menor que 44 y mayor que 11, sitúandose a la derecha de 11. Al ingresar el dato 60, 60 es mayor que 50 y menor que 83, quedando a su izquierda. Con relación a 94, 94 es mayor que 50, 83 y 84, posicionándose en el 95 y al ser menor que 95 se coloca a su izquierda. En 29, 29 es menor que 50, mayor que 7, menor que 44, mayor que 11 y menor que 36, ubicándose a la izquierda de 36. En el dato 37, 37 es menor que 50, mayor que 7, menor que 44, mayor que 11 y 36, tomando la derecha de 36.

Solución del ejemplo 6.8

La estructura de datos lograda hasta ahora es:



Solución del ejemplo 6.8

En el grafo anterior, el estudiante puede inferir que los lados 44 ●–● 11 y 95 ●–● 94 tienen una dirección a la izquierda y las aristas 11 ●–● 36 y 84 ●–● 95 una dirección a la derecha.

Con respecto a las datos restantes, la siguiente tabla resume la manera en como cada uno se coloca en el árbol binario:

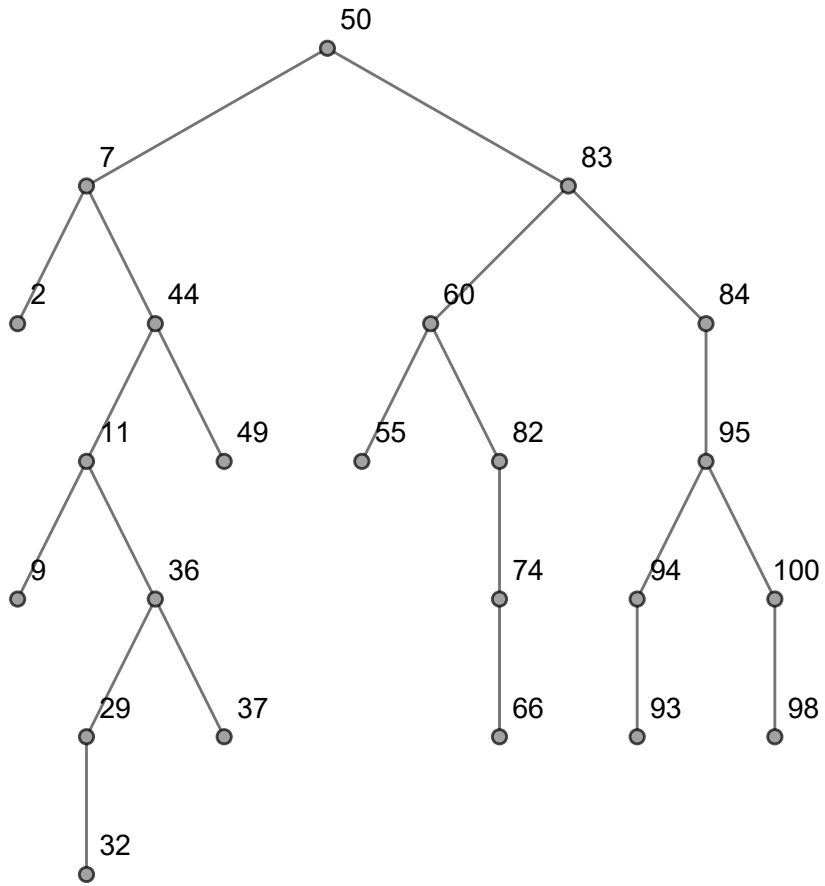
Solución del ejemplo 6.8

Dato	Análisis	Ubicación
100	$100 > 50$, $100 > 83$, $100 > 84$ y $100 > 95$	A la derecha de 95
82	$82 > 50$, $82 < 83$ y $82 > 60$	A la derecha de 60
55	$55 > 50$, $55 < 83$ y $55 < 60$	A la izquierda de 60
9	$9 < 50$, $9 > 7$, $9 < 44$ y $9 < 11$	A la izquierda de 11
32	$32 < 50$, $32 > 7$, $32 < 44$, $32 > 11$, $32 < 36$ y $32 > 29$	A la derecha de 29
93	$93 > 50$, $93 > 83$, $93 > 84$, $93 < 95$ y $93 < 94$	A la izquierda de 94
74	$74 > 50$, $74 < 83$, $74 > 60$ y $74 < 82$	A la izquierda de 82
49	$49 < 50$, $49 > 7$ y $49 > 44$	A la derecha de 44
98	$98 > 50$, $98 > 83$, $98 > 84$, $98 > 95$ y $98 < 100$	A la izquierda de 100
66	$66 > 50$, $66 < 83$, $66 > 60$, $66 < 82$ y $66 < 74$	A la izquierda de 74

(26)

Solución del ejemplo 6.8

Finalmente, se obtiene el árbol binario de búsqueda:



Solución del ejemplo 6.8

En este resultado, se reitera como ya se mencionó en la tabla 26, que las aristas 82 ●–● 74, 74 ●–● 66, 94 ●–● 93 y 100 ●–● 98 tienen una dirección a la izquieda mientras que el lado 29 ●–● 32 posee una dirección a la derecha.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-144.zip

Example (6.9)

Almacene en un árbol binario de búsqueda los datos de:

$$Datos = \{86, 3, 52, 98, 14, 38, 24, 1, 61, 48, 57, 25, 70, 34, 74, 72, 85, 76,$$

$$50, 54, 100, 4, 12\}$$

Solución del ejemplo 6.9

Dato	Análisis	Ubicación
86	Ninguno	Es la raíz del árbol
3	$3 < 86$	A la izquierda de 86
52	$52 < 86$ y $52 > 3$	A la derecha de 3
98	$98 > 86$	A la derecha de 86
14	$14 < 86$, $14 > 3$ y $14 < 52$	A la izquierda de 52
38	$38 < 86$, $38 > 3$, $38 < 52$ y $38 > 14$	A la derecha de 14
24	$24 < 86$, $24 > 3$, $24 < 52$, $24 > 14$ y $24 < 38$	A la izquierda de 38
1	$1 < 86$ y $1 < 3$	A la izquierda de 3
61	$61 < 86$, $61 > 3$ y $61 > 52$	A la derecha de 52
48	$48 < 86$, $48 > 3$, $48 < 52$, $48 > 14$ y $48 > 38$	A la derecha de 38

Solución del ejemplo 6.9

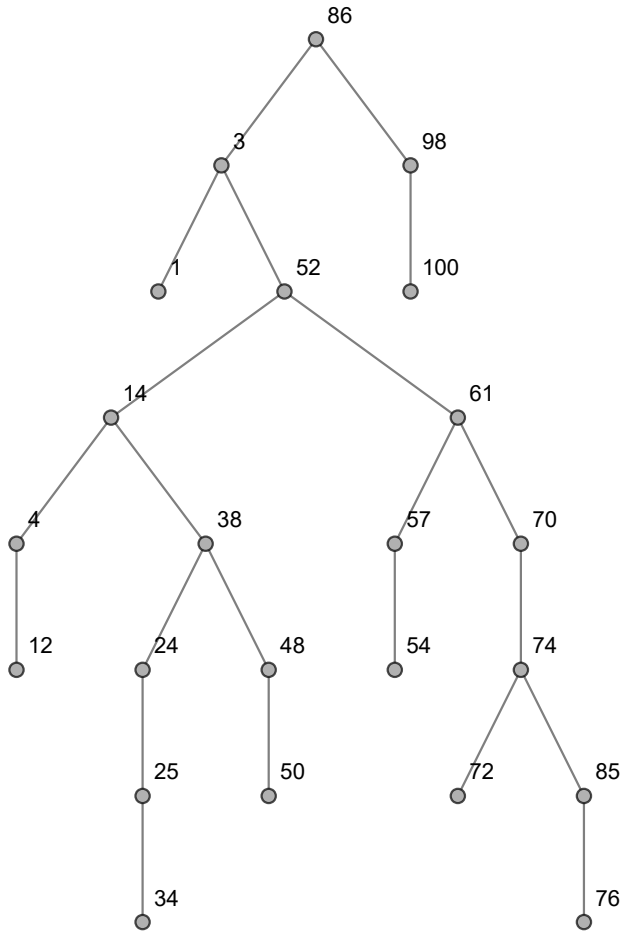
57	$57 < 86$, $57 > 3$, $57 > 52$ y $57 < 61$	A la izquierda de 61
25	$25 < 86$, $25 > 3$, $25 < 52$, $25 > 14$, $25 < 38$ y $25 > 24$	A la derecha de 24
70	$70 < 86$, $70 > 3$, $70 > 52$ y $70 > 61$	A la derecha de 61
34	$34 < 86$, $34 > 3$, $34 < 52$, $34 > 14$, $34 < 38$, $34 > 24$ y $34 > 25$	A la derecha de 25
74	$74 < 86$, $74 > 3$, $74 > 52$, $74 > 61$ y $74 > 70$	A la derecha de 70
72	$72 < 86$, $72 > 3$, $72 > 52$, $72 > 61$, $72 > 70$ y $72 < 74$	A la izquierda de 74

Solución del ejemplo 6.9

85	$85 < 86$, $85 > 3$, $85 > 52$, $85 > 61$, $85 > 70$ y $85 > 74$	A la derecha de 74
76	$76 < 86$, $76 > 3$, $76 > 52$, $76 > 61$, $76 > 70$, $76 > 74$ y $76 < 85$	A la izquierda de 85
50	$50 < 86$, $50 > 3$, $50 < 52$, $50 > 14$, $50 > 38$ y $50 > 48$	A la derecha de 48
54	$54 < 86$, $54 > 3$, $54 > 52$, $54 < 61$ y $54 < 57$	A la izquierda de 57
100	$100 > 86$ y $100 > 98$	A la derecha de 98
4	$4 < 86$, $4 > 3$, $4 < 52$ y $4 < 14$	A la izquierda de 14
12	$12 < 86$, $12 > 3$, $12 < 52$, $12 < 14$ y $12 > 4$	A la derecha de 4

Solución del ejemplo 6.9

Finalmente, el árbol binario de búsqueda corresponde a:



(27)

Solución del ejemplo 6.9

En *Mathematica* la construcción de este tipo de árboles se puede automatizar empleando la sentencia `ArbolBinarioBusqueda` del paquete **VilCretas**. En este ejercicio:

In[ ] :=

```
ArbolBinarioBusqueda[{86, 3, 52, 98, 14, 38, 24, 1, 61, 48,
57, 25, 70, 34, 74, 72, 85, 76, 50, 54, 100, 4, 12}]
```

El **In[ ]** ofrece como salida el árbol compartido en 27.

Nota

`ArbolBinarioBusqueda` al recibir como argumento una lista que contiene datos repetidos, los elimina de forma automática y genera el árbol binario de búsqueda bajo esa condición.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-145.zip

Explicación en video

https://youtu.be/VpMfMWocPmI

Example (6.10)

Diseñe un árbol binario de búsqueda cuyos vértices contengan los elementos de:

$$Datos = \{p, e, y, k, c, t, b, j, u, f, i, d, m, x, l, v, n, a, g\}$$

Solución del ejemplo 6.10

La lista *Datos* a diferencia de los dos ejemplos anteriores, está compuesta por letras del abecedario. Aún así, el algoritmo de elaboración de un árbol binario de búsqueda es aplicable en este ejercicio, asociando a cada letra del alfabeto un número natural consecutivo iniciando en 1 y posteriomente sustituyendo de manera temporal, cada letra de *Datos* por el número correspondiente. En *Wolfram Mathematica* al correr el siguiente código, se automatiza la asignación de las letras del abecedario a un número entero positivo:

In[ ] :=

```
Table[Alphabet["Spanish"][[valor]] -> valor, {valor, 27}]
```

Out[ ] =

{a-> 1, b-> 2, c-> 3, d-> 4, e-> 5, f-> 6, g-> 7, h-> 8, i-> 9, j-> 10, k-> 11, l-> 12, m-> 13, n-> 14, ñ-> 15, o-> 16, p-> 17, q-> 18, r-> 19, s-> 20, t-> 21, u-> 22, v-> 23, w-> 24, x-> 25, y-> 26, z-> 27}

Solución del ejemplo 6.10

`Alphabet` es un comando nativo de *Mathematica* que está devolviendo en el **Out[ ]** cada una de las letras del abecedario en el idioma español. Luego, al cambiar las letras de *Datos* por los números señalados en la salida, se obtiene:

$$Datos = \{17, 5, 26, 11, 3, 21, 2, 10, 22, 6, 9, 4, 13, 25, 12, 23, 14, 1, 7\}$$

El ejercicio se prosigue de la forma acostumbrada:

Solución del ejemplo 6.10

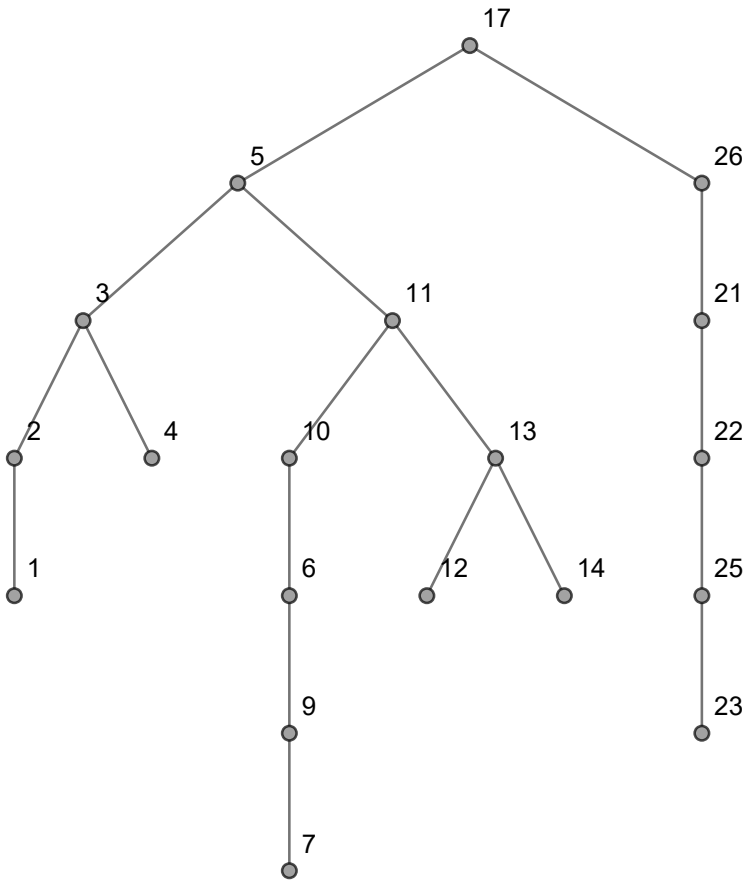
Dato	Análisis	Ubicación
17	Ninguno	Es la raíz del árbol
5	$5 < 17$	A la izquierda de 17
26	$26 > 17$	A la derecha de 17
11	$11 < 17$ y $11 > 5$	A la derecha de 5
3	$3 < 17$ y $3 < 5$	A la izquierda de 5
21	$21 > 17$ y $21 < 26$	A la izquierda de 26
2	$2 < 17$, $2 < 5$ y $2 < 3$	A la izquierda de 3
10	$10 < 17$, $10 > 5$ y $10 < 11$	A la izquierda de 11
22	$22 > 17$, $22 < 26$ y $22 > 21$	A la derecha de 21
6	$6 < 17$, $6 > 5$, $6 < 11$ y $6 < 10$	A la izquierda de 10
9	$9 < 17$, $9 > 5$, $9 < 11$, $9 < 10$ y $9 > 6$	A la derecha de 6

Solución del ejemplo 6.10

4	$4 < 17$, $4 < 5$ y $4 > 3$	A la derecha de 3
13	$13 < 17$, $13 > 5$ y $13 > 11$	A la derecha de 11
25	$25 > 17$, $25 < 26$, $25 > 21$ y $25 > 22$	A la derecha de 22
12	$12 < 17$, $12 > 5$, $12 > 11$, y $12 < 13$	A la izquierda de 13
23	$23 > 17$, $23 < 26$, $23 > 21$, $23 > 22$ y $23 < 25$	A la izquierda de 25
14	$14 < 17$, $14 > 5$, $14 > 11$, y $14 > 13$	A la derecha de 13
1	$1 < 17$, $1 < 5$, $1 < 3$ y $1 < 2$	A la izquierda de 2
7	$7 < 17$, $7 > 5$, $7 < 11$, $7 < 10$, $7 > 6$ y $7 < 9$	A la izquierda de 9

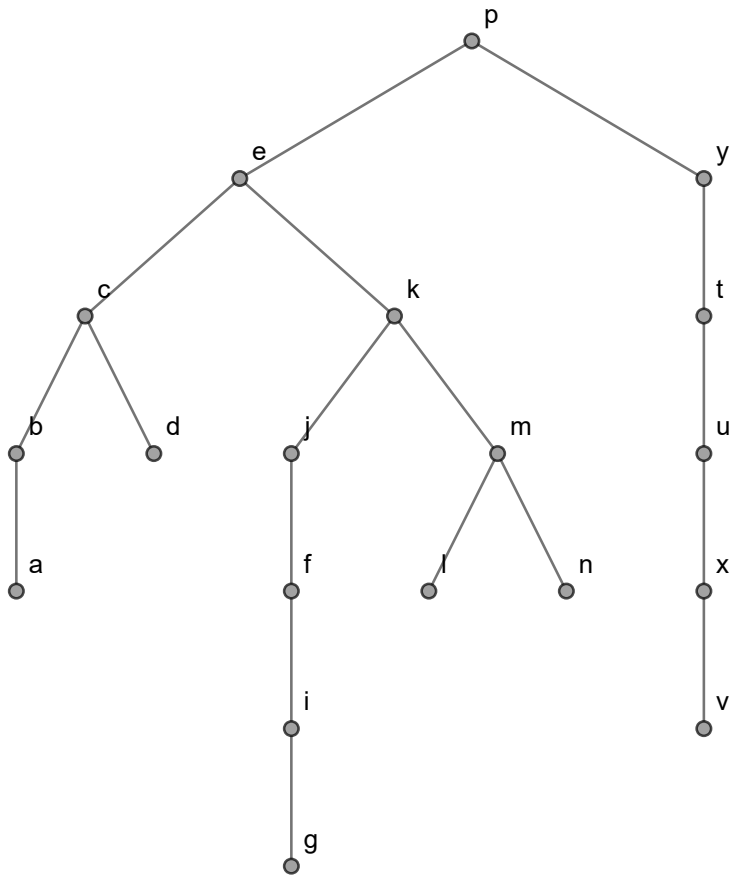
Solución del ejemplo 6.10

En consecuencia, el árbol binario de búsqueda es:



Solución del ejemplo 6.10

Finalmente, en este grafo se sustituye cada número entero positivo por la letra vinculada con él:



(28)

Solución del ejemplo 6.10

La instrucción `ArbolBinarioBusqueda` también elabora la estructura de datos, aún cuando la lista pasada como argumento, sea un conjunto de caracteres como sucede en este ejemplo. El **In[ ]** compartido a continuación retorna el árbol binario de búsqueda 28:

In[ ] :=

```
ArbolBinarioBusqueda[{p, e, y, k, c, t, b, j, u, f, i, d,
m, x, l, v, n, a, g}]
```

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-146.zip

Example (6.11)

Genere un árbol binario de búsqueda para la frase del célebre físico teórico *Stephen Hawking*: "el universo no se comporta de acuerdo con nuestras ideas preconcebidas".

Solución del ejemplo 6.11

Si se desea hallar un árbol binario de búsqueda para la frase del enunciado, se inicia extrayendo cada palabra de la frase y colocándo en orden de aparición, los *strings* en un conjunto *Datos*. Luego:

$$Datos = \{\text{el, universo, no, se, comporta, de, acuerdo, con, nuestras, ideas, preconcebidas}\}$$

Se insiste que el árbol existirá siempre y cuando *Datos* no contenga elementos repetidos.
La construcción del árbol binario de búsqueda se realiza tal y como se explicó en el ejemplo 14, seleccionando para ello, la primera letra de la palabra a insertar y el primer caracter del *string* a comparar. Si esta primera letra coincide en ambos casos entonces se elige el segundo caracter de ambas palabras para realizar el proceso, si a su vez, esta segunda letra es igual, se utiliza el tercer caracter de los *strings* y así sucesivamente.

Solución del ejemplo 6.11

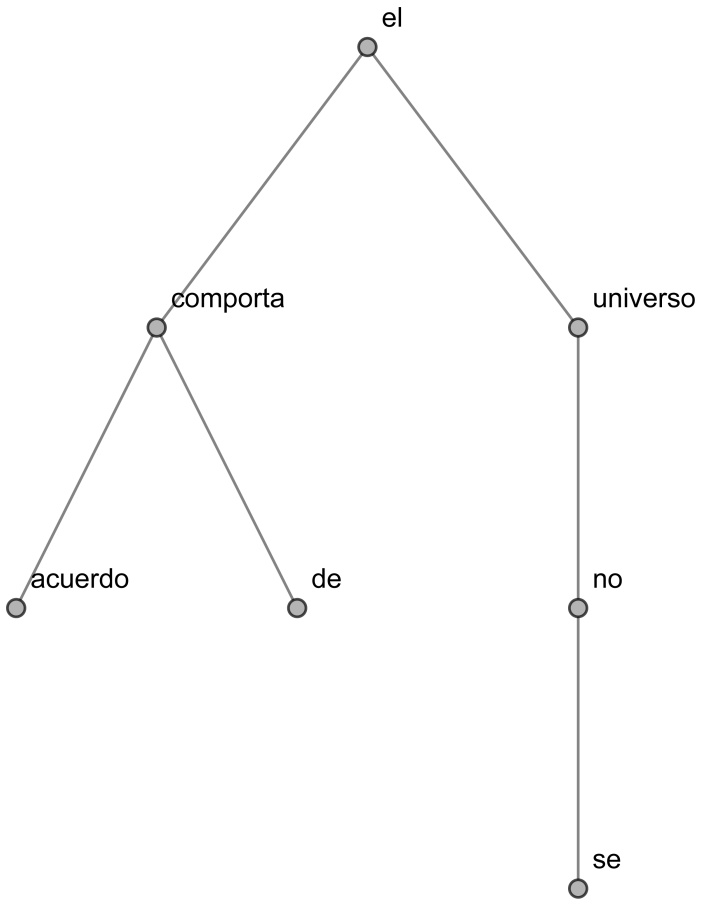
De acuerdo con lo anterior, se inicia colocando la palabra "el" en la raíz del árbol binario de búsqueda. Ahora se inserta el *string* "universo" comparando la "u" equivalente al número entero 22, con la letra "e" asociada al número 5. Como $22 > 5$, se ubica "universo" a la derecha de "el". Luego, al tomar a "no", se compara la letra "n" (vinculada con el valor 14), con la letra "e" (equivalente a 5), $14 > 5$, "no" va a la derecha de "el" y como $14 < 22$, "no" se sitúa a la izquierda de "universo". Con relación al *string* "se" su primera letra "s" se vincula con el número entero 20, donde $20 > 5$, $20 < 22$ y $20 > 14$, indicando que "se" ocupa una posición a la derecha de "no".

Solución del ejemplo 6.11

Por otra parte, "comporta" se asocia al entero 3 por su primera letra igual a "c", $3 < 5$, se ubica "comporta" a la izquierda de "el". El siguiente *string* a insertar es "de" relacionado con el valor 4, $4 < 5$ y $4 > 3$, implicando la necesidad de colocar "de" a la derecha de "comporta". Luego, "acuerdo" contiene como su primera letra la "a", razón por la cual, se vincula con el número 1, $1 < 5$ y $1 < 3$, en consecuencia, "acuerdo" se sitúa a la izquierda de "comporta".

Solución del ejemplo 6.11

Gráficamente:



Solución del ejemplo 6.11

En la palabra "con" ocurre una coincidencia de caracteres en el proceso de inserción. El *string* "con" se relaciona al entero 3 (por la "c" con la cual inicia), 3 es menor que 5, provocando un traslado de "con" a la izquierda de la raíz "el", donde se ubica la palabra "comporta". Los *strings* "con" y "comporta" tienen sus dos primeros caracteres iguales, a razón de ello, la comparación entre ambas palabras se efectúa al extraer las letras ubicadas en la tercera posición, es decir, se toma el caracter "n" de "con" equivalente a 14 y la letra "m" de "comporta" asociada al número 13, $14 > 13$ y en consecuencia, "con" manifiesta una dirección a la derecha de "comporta", donde está almacenado el *string* "de".

Solución del ejemplo 6.11

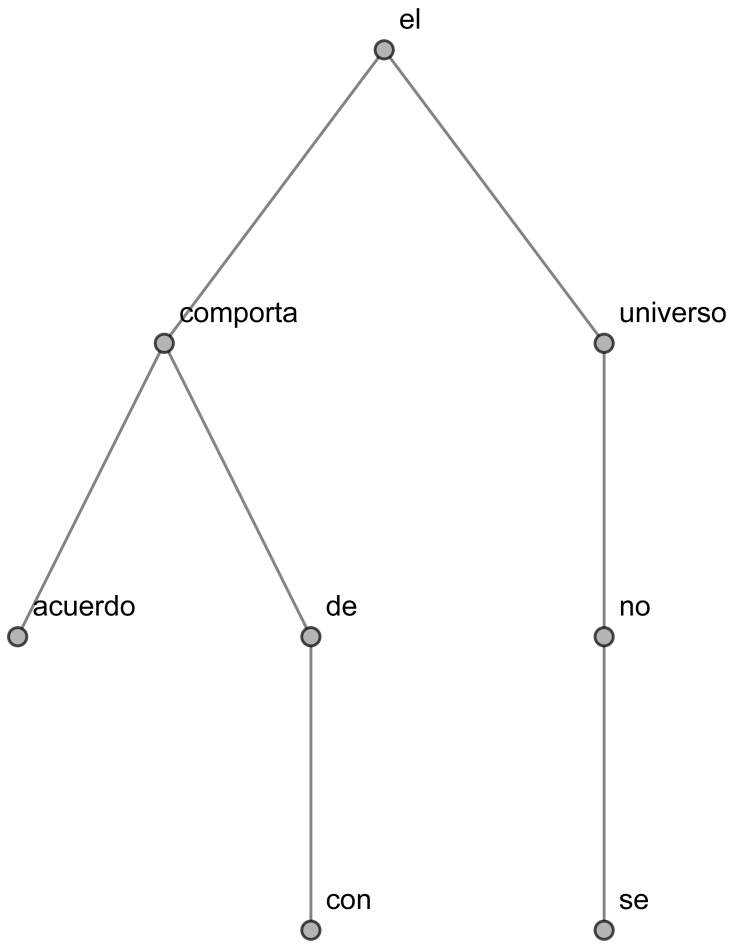
Allí, al tomar la primera letra "c" de "con" y el primer caracter "d" de la palabra "de", se comparan usando sus equivalentes numéricos 3 y 4, respectivamente. Como $3 < 4$, "con" toma un dirección a la izquierda de "de".

Nota

El alumno debe tener claro, en este sentido, que el movimiento a otro caracter distinto del primero ocurre en la operación de inserción, únicamente cuando las palabras tienen la misma o las mismas letras iniciales. Si los primeros caracteres de los *strings* comparados son distintos, no se debe realizar la inserción respectiva, usando otras de las letras subsiguientes.

Solución del ejemplo 6.11

El árbol en este punto, toma la forma:



Solución del ejemplo 6.11

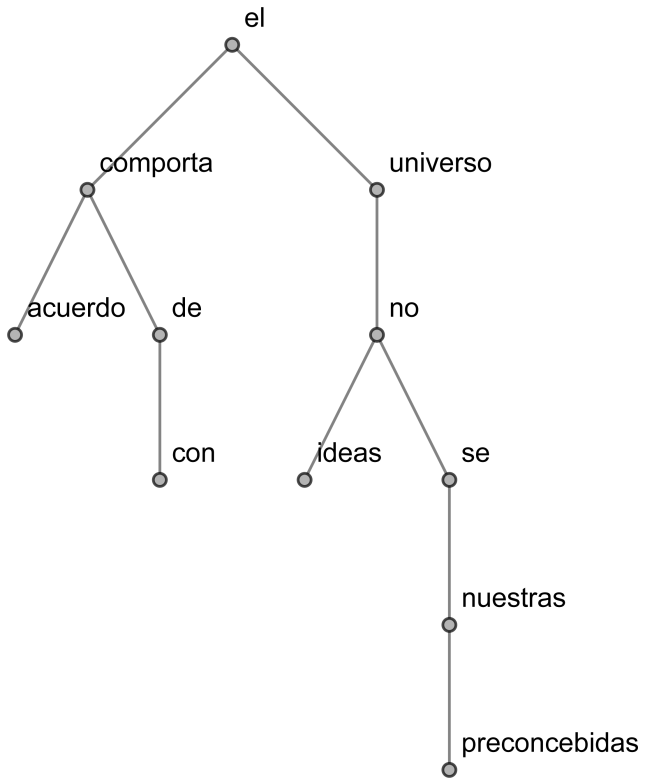
Ahora se inserta la palabra "nuestras". El primer caracter "n" se relaciona con el número entero 14. En consecuencia, $14 > 5$ y $14 < 22$, llegando al vértice que contiene la palabra "no". Los *strings* "nuestras" y "no" comparten la "n" como su primer caracter, por lo tanto, se extraen sus segundos caracteres "u" (vinculado con 22) y "o" (asociado con 16), respectivamente. Aquí, $22 > 16$, "nuestras" se ubica a la derecha de "no" donde se encuentra la palabra "se". Nuevamente, "nuestras" se relaciona con el entero 14 y el *string* "se" con el valor 20, $14 < 20$, por lo que, "nuestras" queda a la izquierda de "se".

Solución del ejemplo 6.11

Corresponde añadir al árbol, la palabra "ideas", siendo la letra "i" equivalente al entero 9, $9 > 5$, $9 < 22$ y $9 < 14$, ubicándose "ideas" a la izquierda de "no". Finalmente, prosigue insertar el *string* "preconcebidas", la "p" corresponde al número 17, $17 > 5$, $17 < 22$, $17 > 14$, $17 < 20$ y $17 > 14$, colocándose "preconcebidas" a la derecha de "nuestras".

Solución del ejemplo 6.11

El árbol binario de búsqueda elaborado para la frase del ejemplo es el siguiente:



Solución del ejemplo 6.11

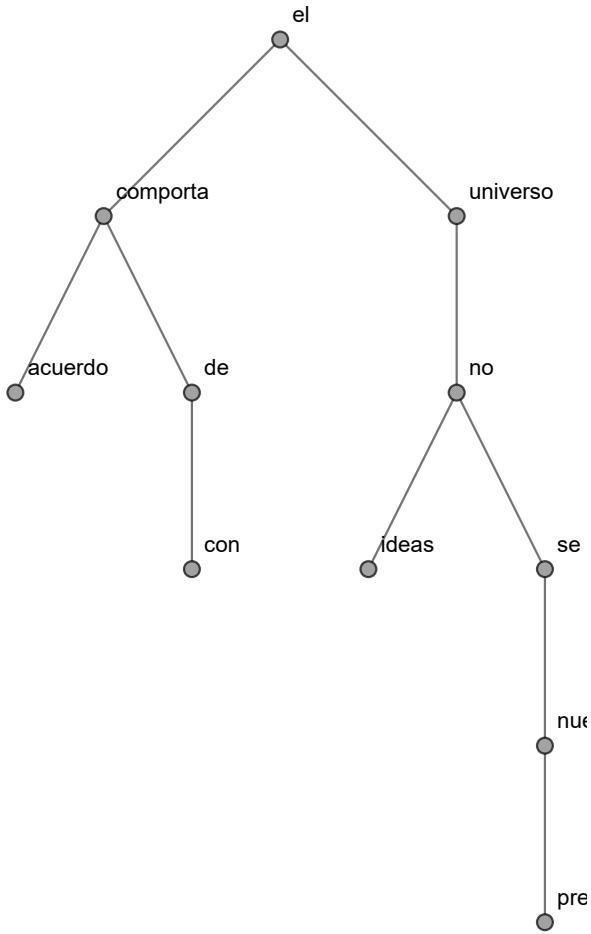
El comando `ArbolBinarioBusqueda`, también acepta como argumento un *string* correspondiente a una frase. En este ejercicio:

In[ ] :=

```
arbol = ArbolBinarioBusqueda["el universo no se comporta de acuerdo con nuestras ideas preconcebidas"]
```

Solución del ejemplo 6.11

Out[ ] =



Solución del ejemplo 6.11

En el árbol de la salida se observan algunos cortes. La palabra "nuestras" y "preconcebidas" no aparecen completas por su longitud, sin embargo, el estudiante puede recurrir a la sentencia `ListaVertices` de la librería **VilCretas** para corroborar la inclusión correcta de todos los *strings* de la frase de interés. En este caso:

In[ ] :=

```
ListaVertices[arbol]
```

Out[ ] =

{acuerdo, comporta, con, de, el, ideas, no, nuestras, preconcebidas, se, universo}

Solución del ejemplo 6.11

El **Out[ ]** anterior, no concuerda en orden con la lista *Datos* original, pues `ListaVertices` por defecto acomoda los elementos de forma alfabética.

Descargue un archivo

`https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-147.zip`

- La siguiente sección aborda una serie de algoritmos reconocidos en la literatura como métodos de trabajo en la ejecución de búsqueda de datos. Estos procedimientos resultan fundamentales para recorrer o visitar los vértices de un árbol binario.

Recorridos en un árbol binario

Prof. Enrique Vílchez Quesada

Universidad Nacional de Costa Rica

Recorridos en un árbol binario

Los recorridos en un árbol binario son formas de desplazamiento sobre cada uno de sus vértices. Un recorrido en un árbol tiene como objetivo pasar una única vez por todos los nodos que lo constituyen. En general, existen tres métodos de recorrido sobre un árbol binario: el orden inicial, preorden o recorrido prefijo, el orden intermedio, interorden o recorrido interfijo y el orden final, postorden o recorrido postfijo. Cada uno se enuncia a continuación como un algoritmo.

Teorema 6.4

Theorem (6.4)

Sea (T, r) un árbol binario con V el conjunto de vértices de T entonces:

1. *El recorrido de orden inicial, preorden o prefijo se describe mediante el siguiente procedimiento:*
   1. *Si V es vacío, regrese.*
   2. *Visite o imprima la raíz r del árbol T.*
   3. *Procese el subárbol que se encuentra a la izquierda de r llamando a este algoritmo.*
   4. *Procese el subárbol que está a la derecha de r y regrese.*

Teorema 6.4

Theorem (6.4. Continuación)

2. *El recorrido de orden intermedio, interorden o interfijo se realiza como sigue:*
   1. *Si V es vacío, regrese.*
   2. *Procese el subárbol que se encuentra a la izquierda de r invocando a este algoritmo.*
   3. *Visite o imprima la raíz r del árbol T.*
   4. *Procese el subárbol que está a la derecha de r y regrese.*
3. *El recorrido de orden final, postorden o postfijo se efectúa así:*
   1. *Si V es vacío, regrese.*
   2. *Procese el subárbol que se encuentra a la izquierda de r a través de este algoritmo.*
   3. *Procese el subárbol que está a la derecha de r.*
   4. *Visite o imprima la raíz r y regrese.*

Comentario sobre el teorema 16

Como se observa los algoritmos preorden, interorden y postorden son métodos recursivos pues cada uno se invoca así mismo. Además, la única diferencia entre ellos, radica en el momento de visitación de la raíz r del árbol T. Si el recorrido es prefijo, r se imprime previamente a cualquier paso. Si el recorrido es interfijo, se visita la raíz r cuando ya se ha procesado todo lo que se encuentra al lado izquierdo. Si el recorrido es postfijo, la impresión se realiza al final.

Los métodos de preorden y postorden son aplicables en árboles no binarios, siendo estas formas de recorrido más generales en comparación con el interorden, pues éste es exclusivo de los árboles binarios. En este libro no se detallará sobre los algoritmos de recorrido prefijo y postfijo en $k-$árboles con $k \neq 2$.

Comentario sobre el teorema 16

Los algoritmos expuestos en el teorema 16 son métodos de búsqueda de datos. En ese contexto, si se desea buscar un dato w sobre un árbol binario T, al recorrer los nodos de T según los distintos órdenes del teorema 16, la búsqueda se realiza comparando w con el vértice visitado, si son iguales se detiene el proceso y se retorna "dato encontrado", en caso contrario se continua con el recorrido respectivo. Si al visitar todos los nodos de T no se encontró a w, se devuelve el mensaje "dato no encontrado". Bajo esta perspectiva, el teorema 16 ofrece tres métodos distintos de búsqueda proporcionando un orden de comparación entre el dato buscado w y la información almacenada en el árbol binario.

- En los ejemplos expuestos a continuación, se explicará cómo realizar los distintos recorridos enunciados en el teorema 16, sin ejecutar ninguna búsqueda de datos. Pese a ello, se insiste al estudiante que estos algoritmos conforman métodos de búsqueda cuando lo que se tiene es una estructura de datos relativa a un árbol de orden 2.

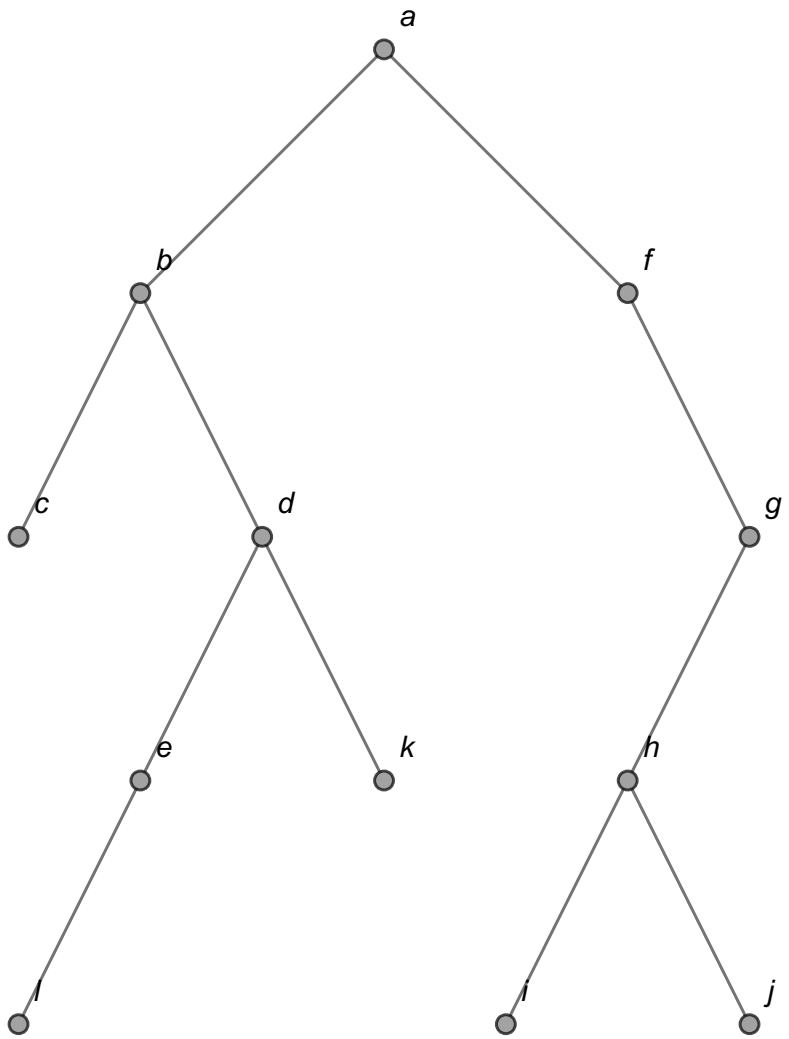


Figura: Árbol del ejemplo 18

Example (6.12)

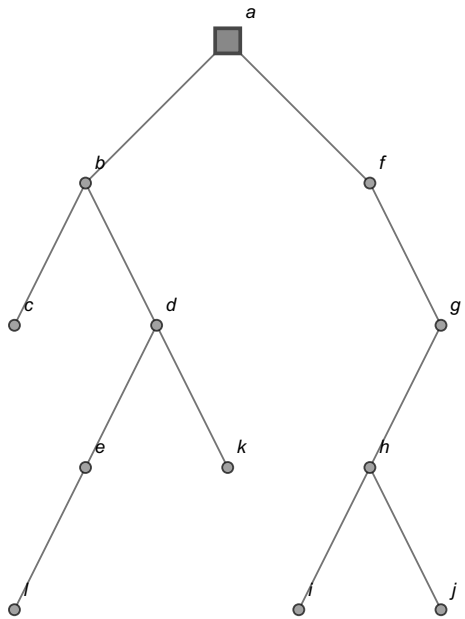
Ejecute los recorridos de orden inicial, intermedio y final en el árbol de la figura 3.

Solución del ejemplo 6.12

Cada uno de los recorridos correspondientes será explicado paso a paso, describiendo las acciones realizadas de acuerdo con el teorema 16. La palabra "visitar" se usará como un término equivalente a "imprimir".

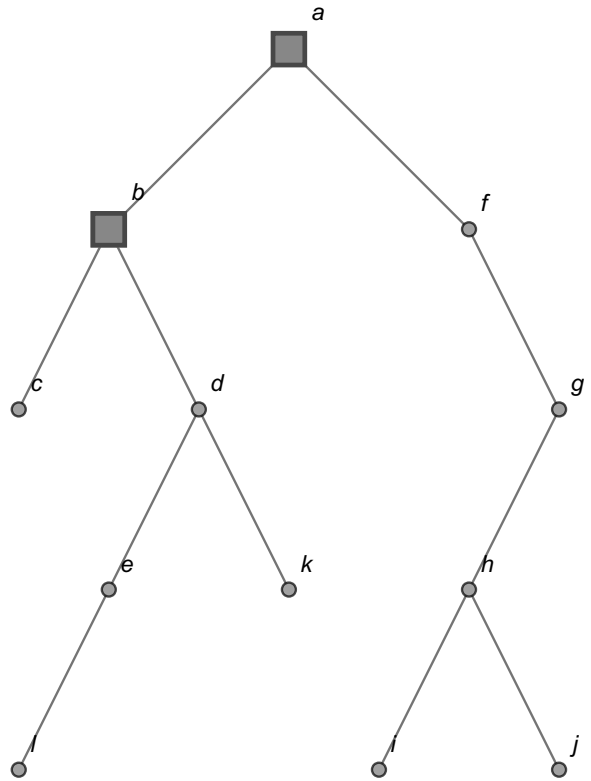
Recorrido de orden inicial:

- Se imprime la raíz *a* :



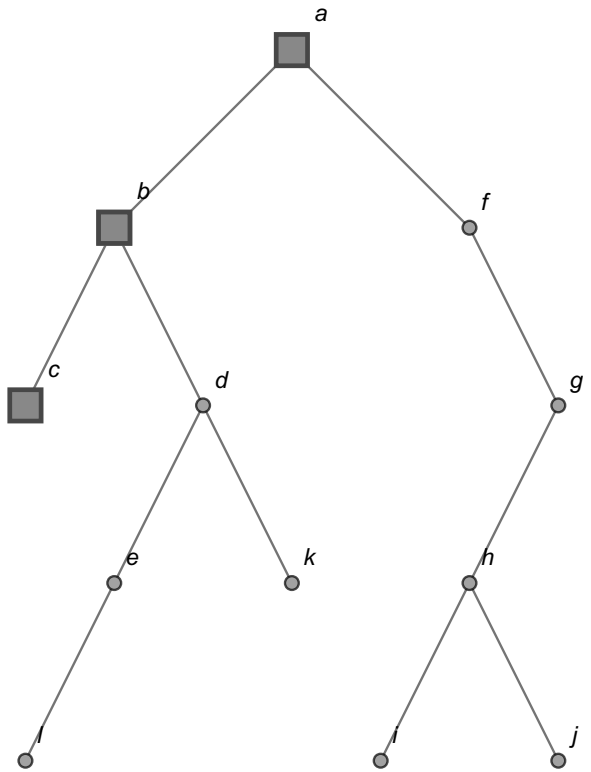
Solución del ejemplo 6.12

- Se pasa a la izquierda de a y se visita b :



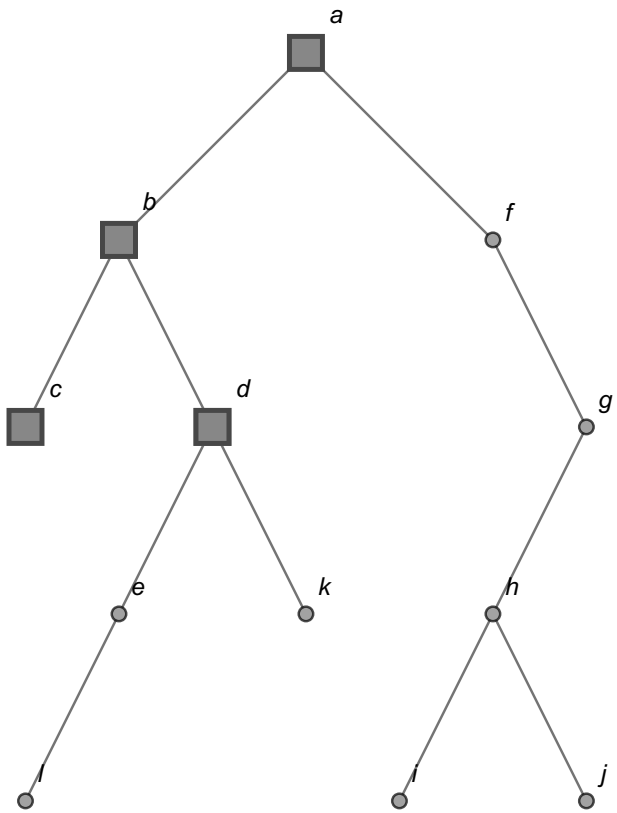
Solución del ejemplo 6.12

- Se realiza un desplazamiento a la izquierda de b y se imprime c :



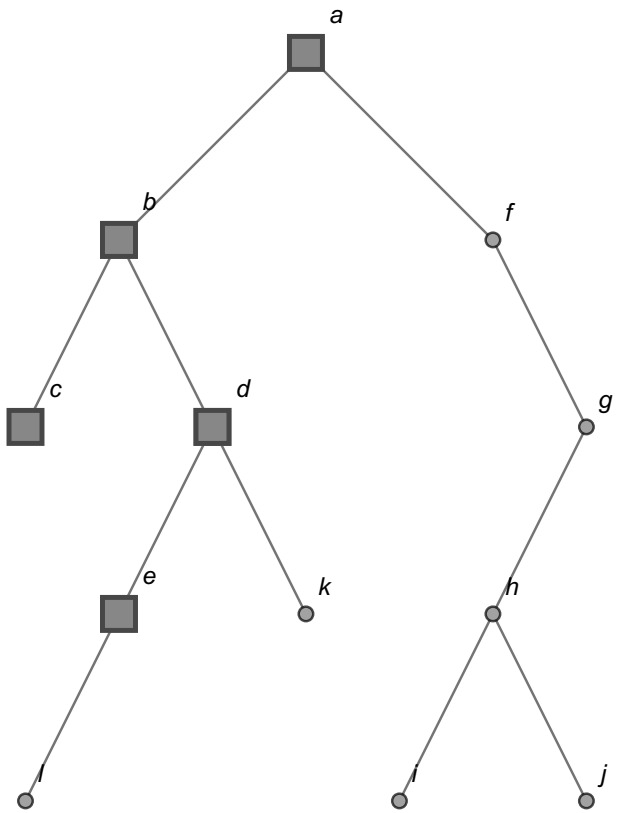
Solución del ejemplo 6.12

- Se regresa a b, se pasa a su derecha y se visita d :



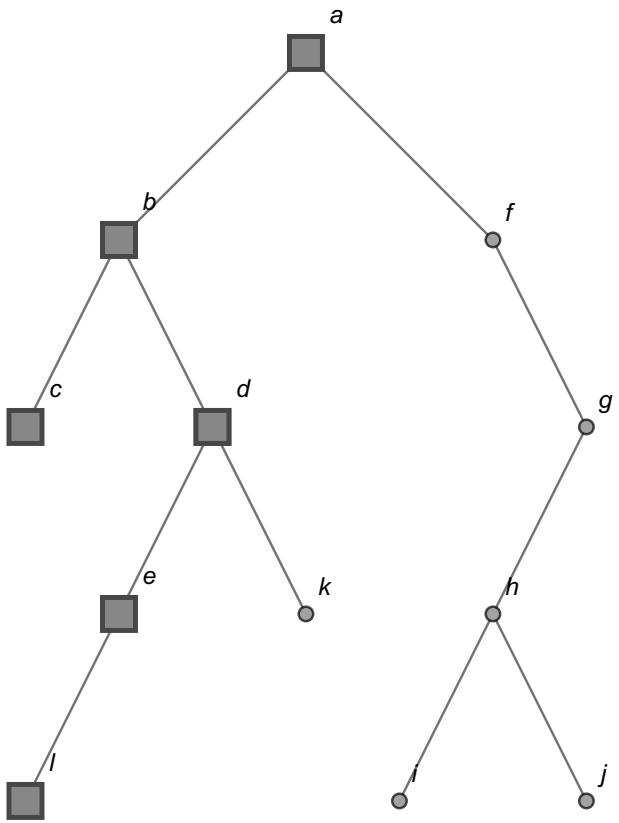
Solución del ejemplo 6.12

- Se pasa a la izquierda de d y se imprime e :



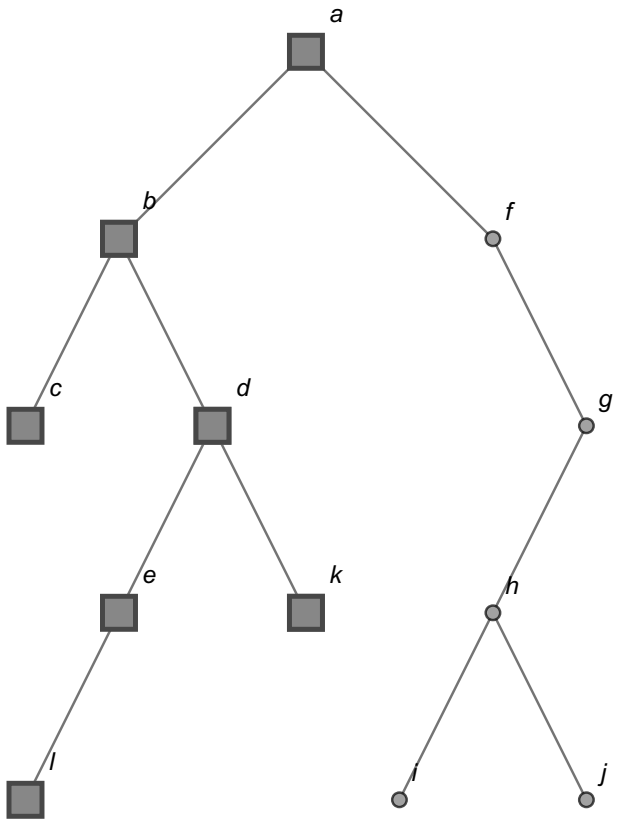
Solución del ejemplo 6.12

- Se realiza un traslado a la izquierda de e y se visita l :



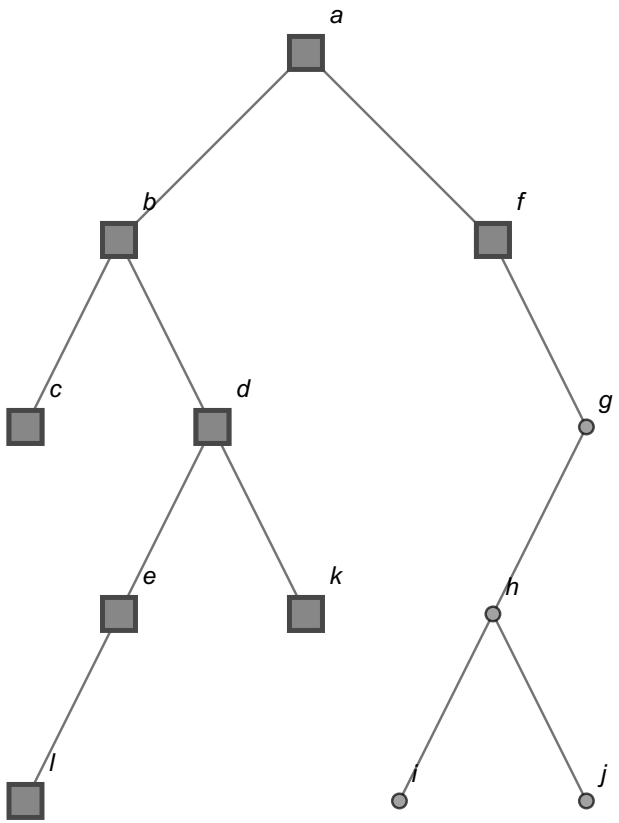
Solución del ejemplo 6.12

- Se regresa a e, como no hay nada a su derecha, se regresa a d, se pasa a su izquierda y se imprime k :



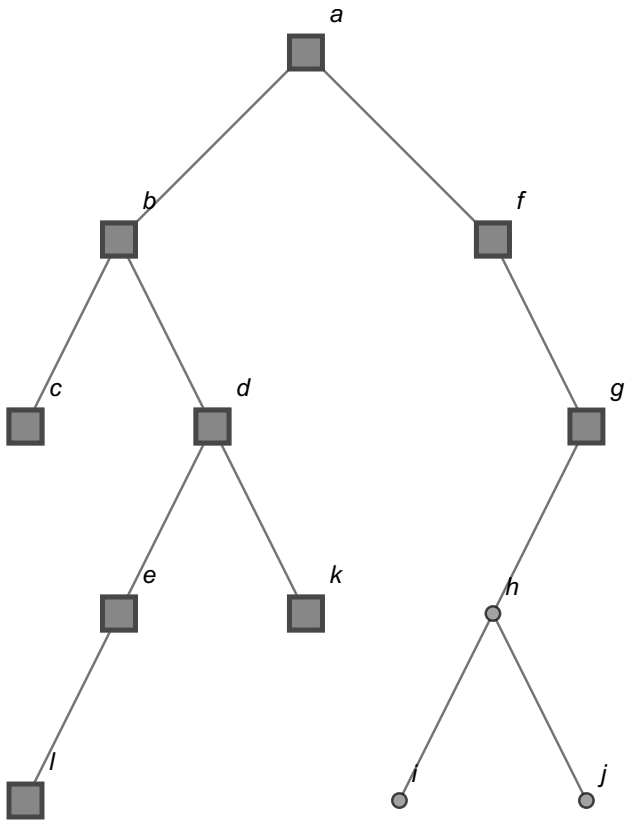
Solución del ejemplo 6.12

- Se regresa a d, b y a. En a se pasa a su derecha y se visita f :



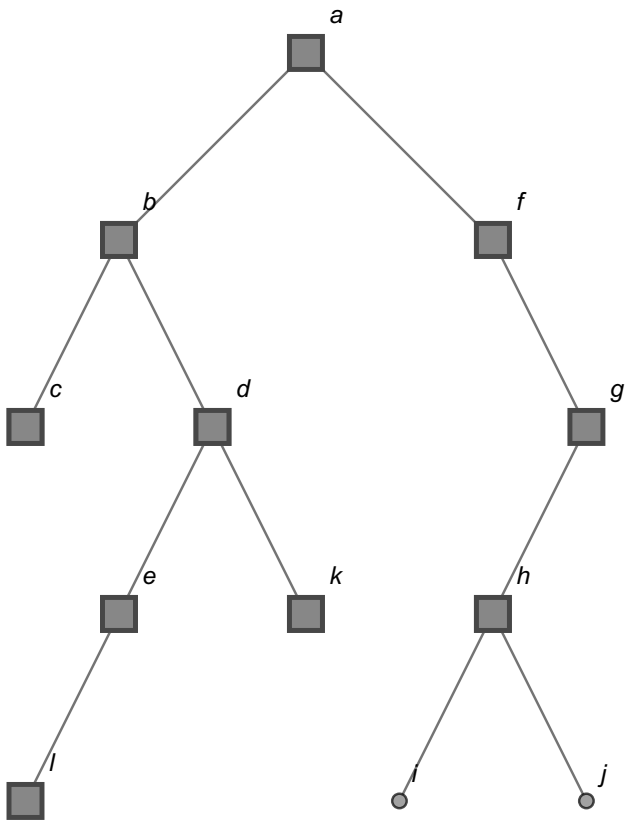
Solución del ejemplo 6.12

- Como en f no hay nada a su izquierda, se pasa a su derecha y se imprime g :



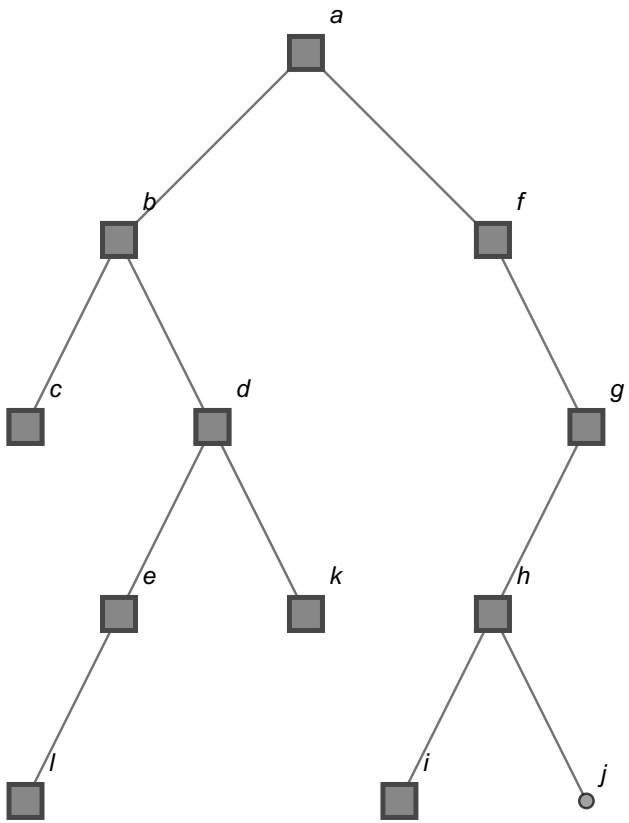
Solución del ejemplo 6.12

- En g se pasa a su izquierda y se imprime h :



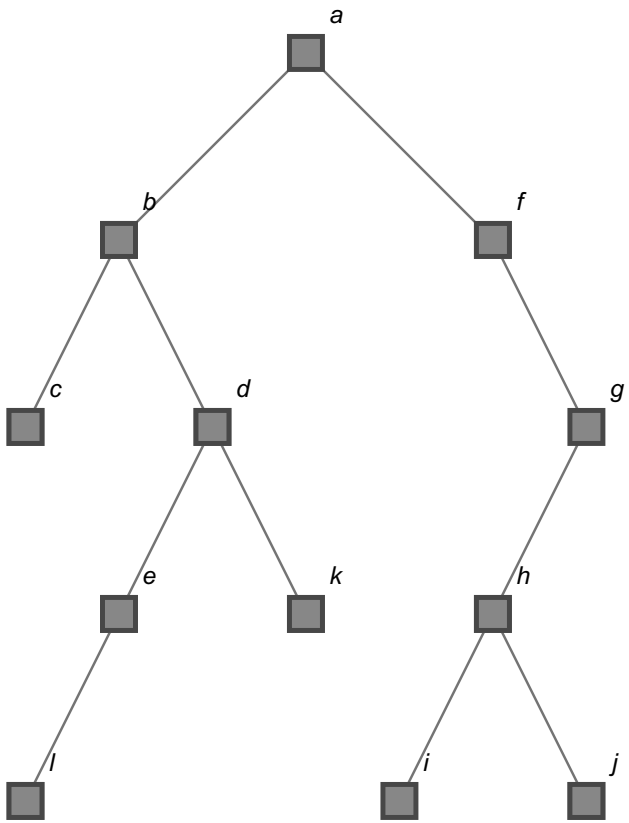
Solución del ejemplo 6.12

- En h se pasa a su izquierda y se visita i :



Solución del ejemplo 6.12

- Se regresa a h, se pasa a su derecha y se imprime j :



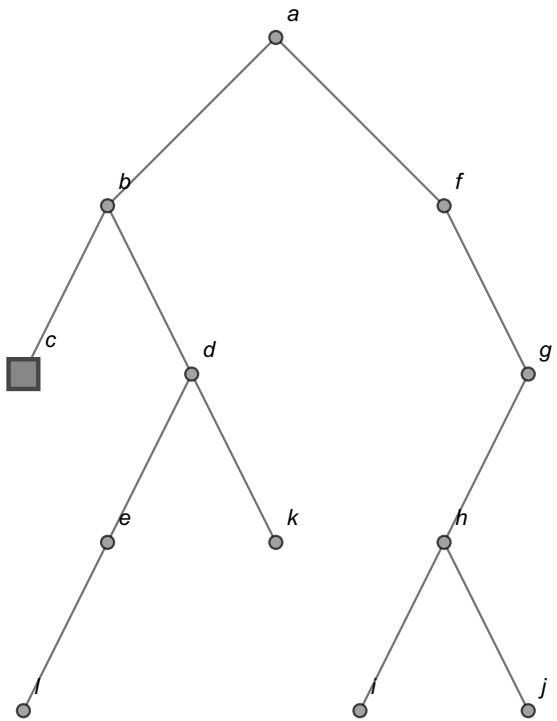
Solución del ejemplo 6.12

- Se regresa a h y g, como no hay nada a la derecha de g, se regresa a f y finalmente se regresa a a. En resumen, el recorrido preorden es: $\{a, b, c, d, e, l, k, f, g, h, i, j\}$.

Solución del ejemplo 6.12

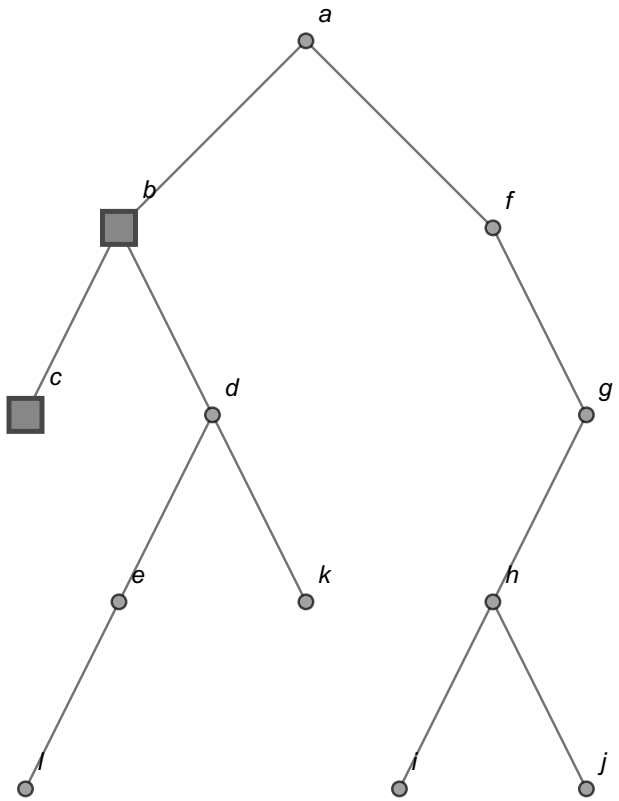
Recorrido de orden intermedio:

- Se parte de la raíz a, se pasa a su izquierda en b, luego a la izquierda de b en c y se imprime c :



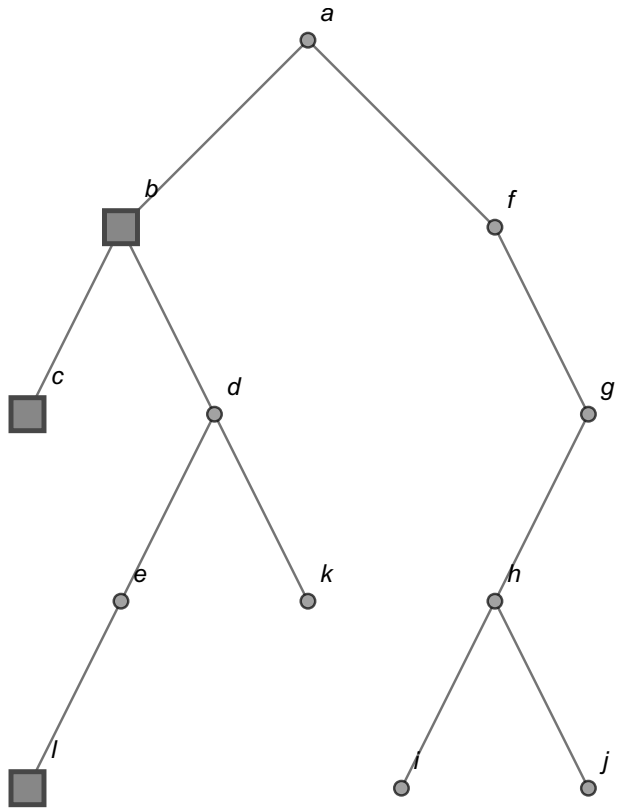
Solución del ejemplo 6.12

- Se regresa a b y se visita b :



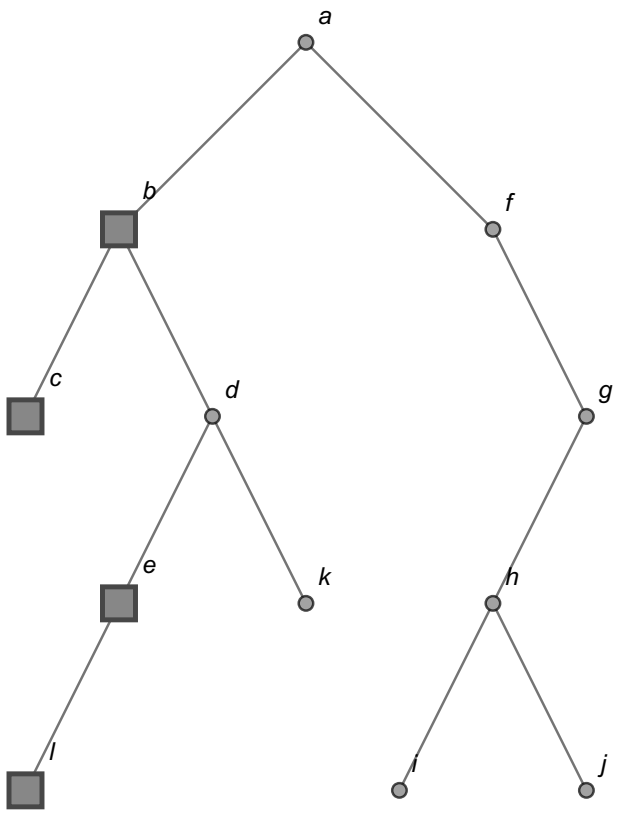
Solución del ejemplo 6.12

- Se pasa a la derecha de b en d. En el nodo d se pasa a su izquierda en e y a la izquierda de e en l, se imprime l :



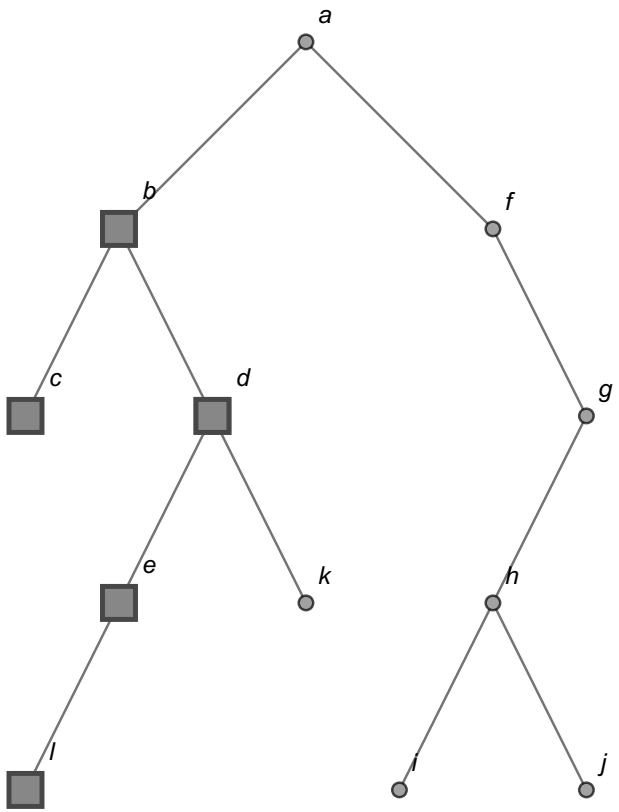
Solución del ejemplo 6.12

- Se regresa a e y se visita e :



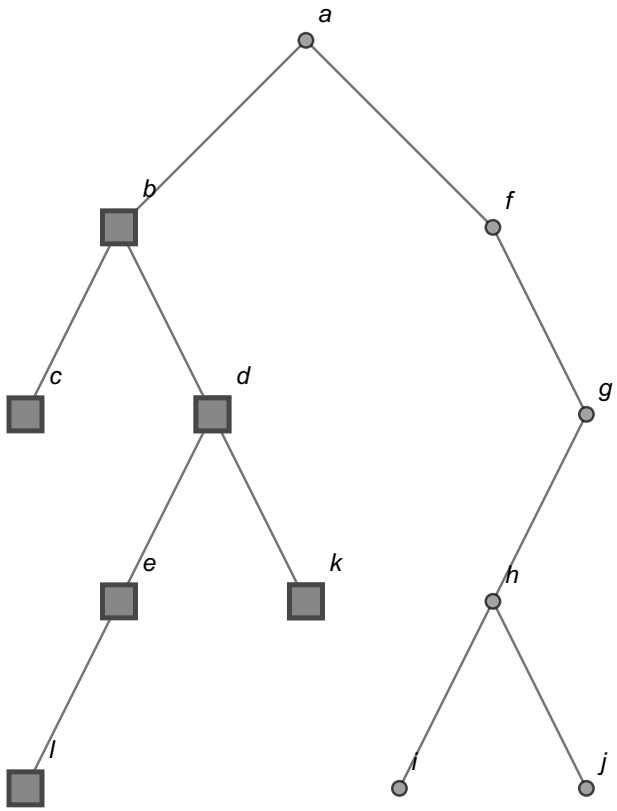
Solución del ejemplo 6.12

- No hay nada a la derecha de e, por lo que, se regresa a d y se imprime d :



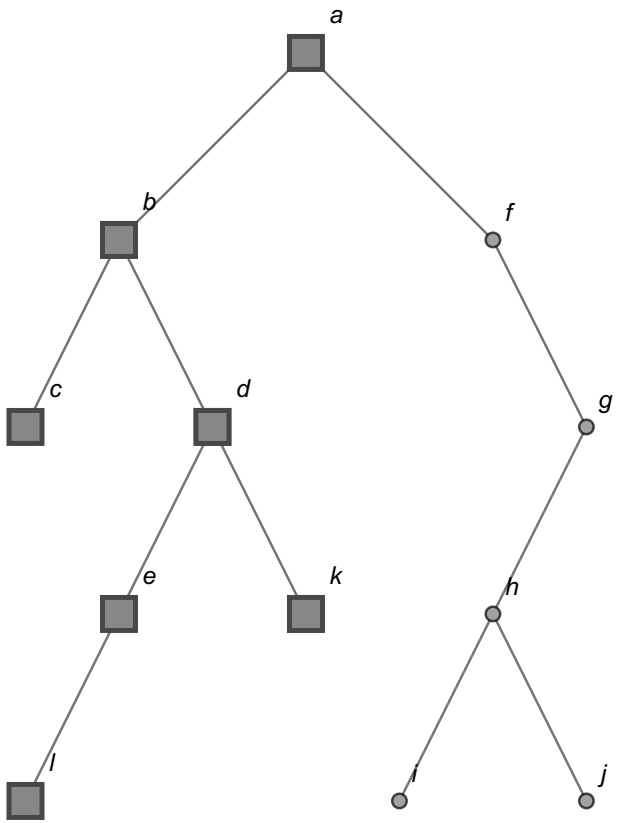
Solución del ejemplo 6.12

- Se pasa a la derecha de d y se visita el nodo k :



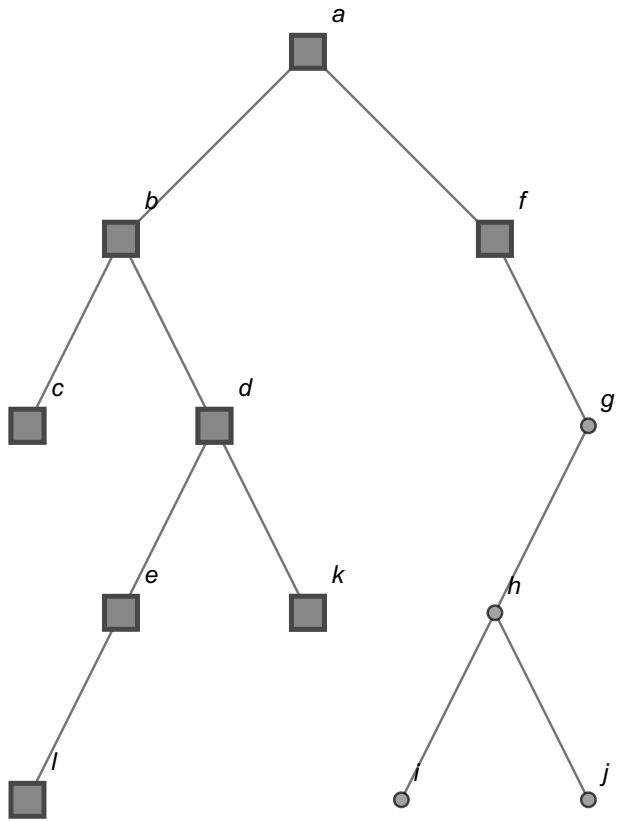
Solución del ejemplo 6.12

- Se regresa a d, b y a, se imprime a :



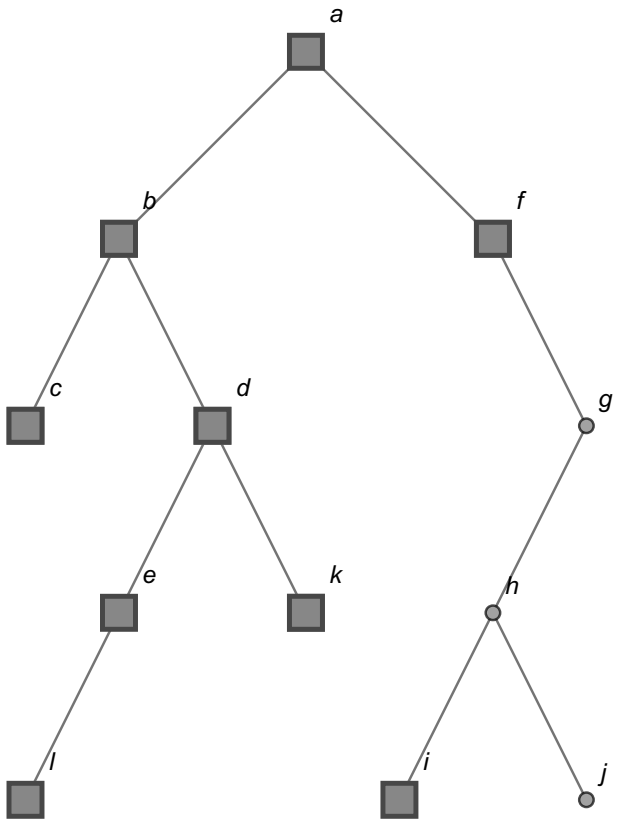
Solución del ejemplo 6.12

- Se pasa a la derecha de a y como en el vértice f no hay nada a la izquierda, se visita f :



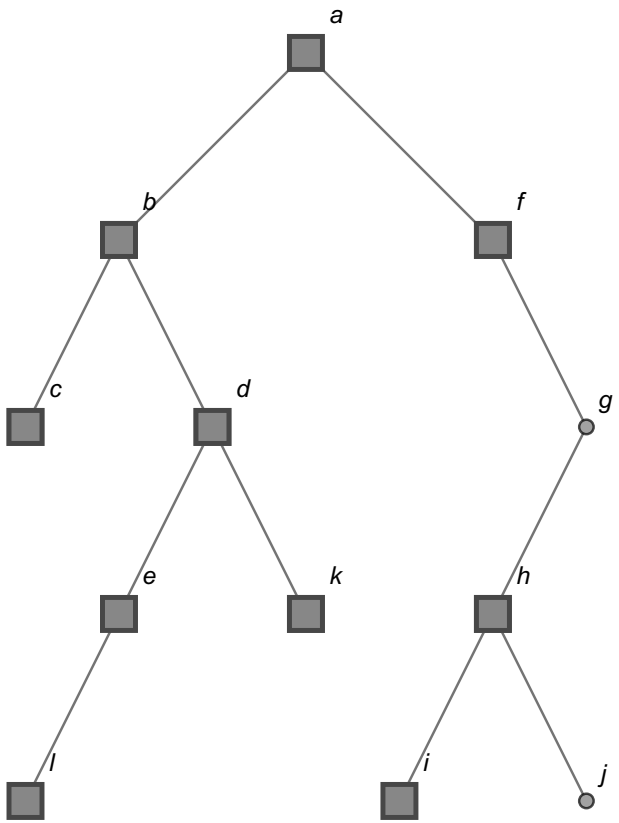
Solución del ejemplo 6.12

- Se realiza un traslado a la derecha de f en g, se pasa a la izquierda de g en h, luego a la izquierda de h en i y se imprime i :



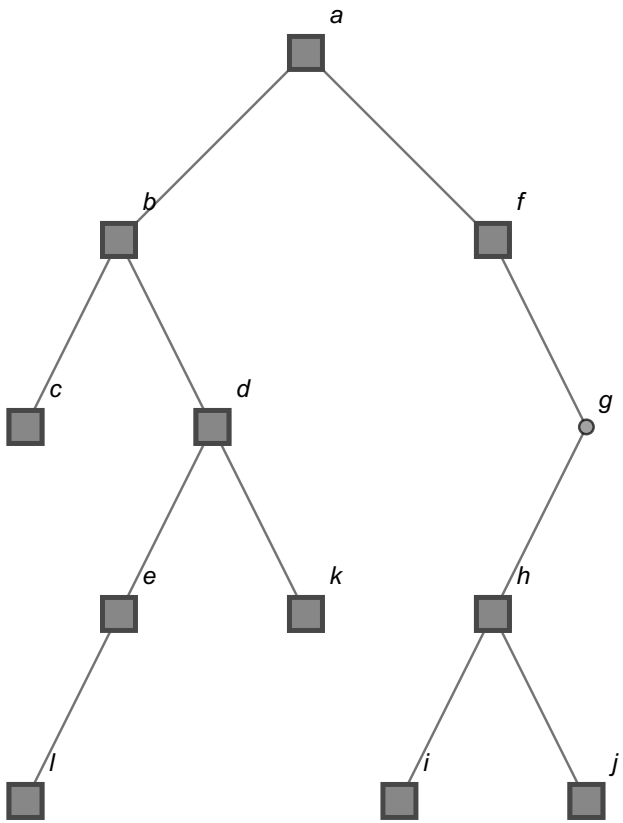
Solución del ejemplo 6.12

- Se regresa a h y se visita h :



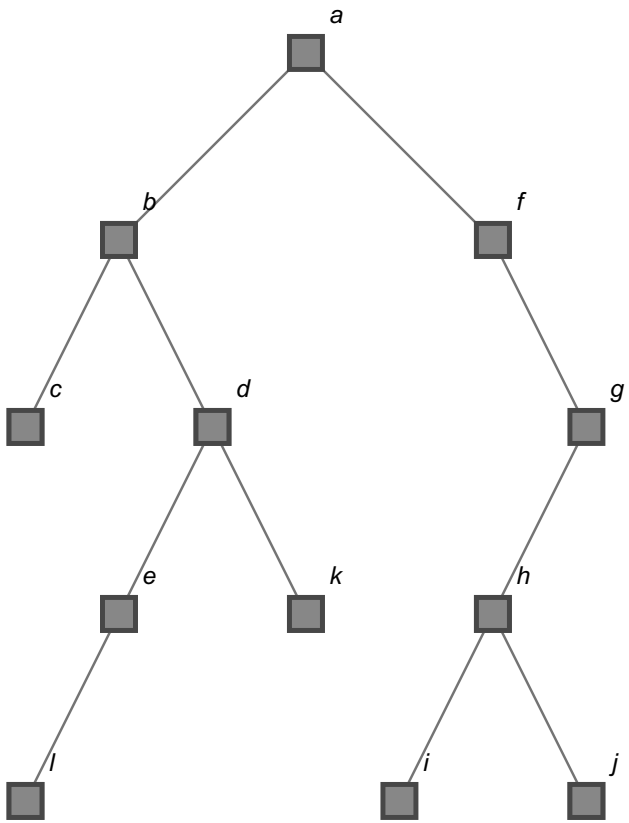
Solución del ejemplo 6.12

- Se pasa a la derecha de h y se imprime j :



Solución del ejemplo 6.12

- Se regresa a h y g, se visita g :



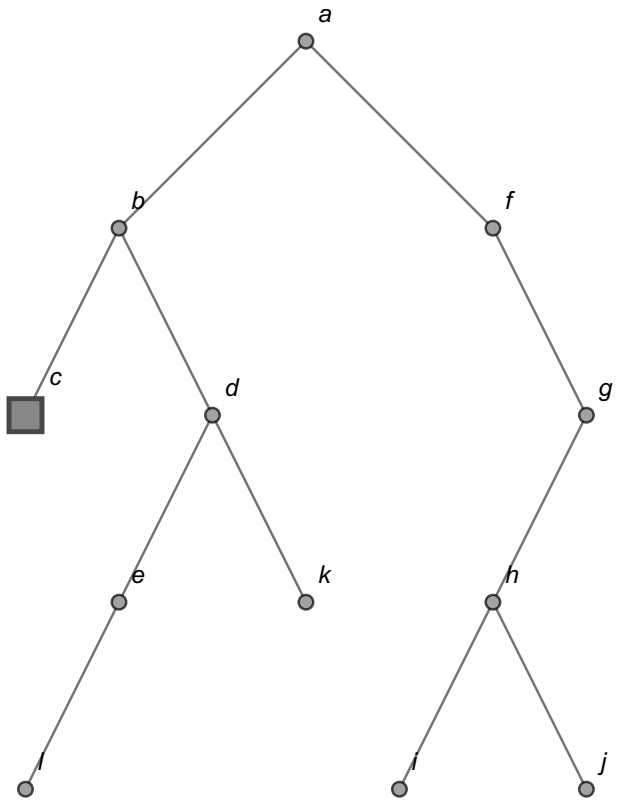
Solución del ejemplo 6.12

- Como en g no hay nada a su derecha, se regresa a f y a. En resumen, el recorrido interorden corresponde a: $\{c, b, l, e, d, k, a, f, i, h, j, g\}$.

Solución del ejemplo 6.12

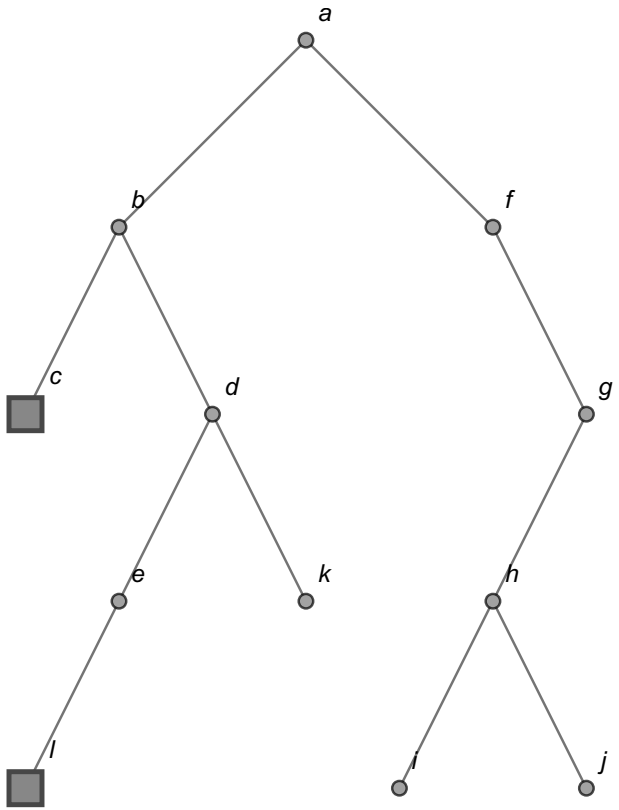
Recorrido de orden final:

- Se inicia en a, se pasa a su izquierda en b y a la izquierda de b en c, se imprime c :



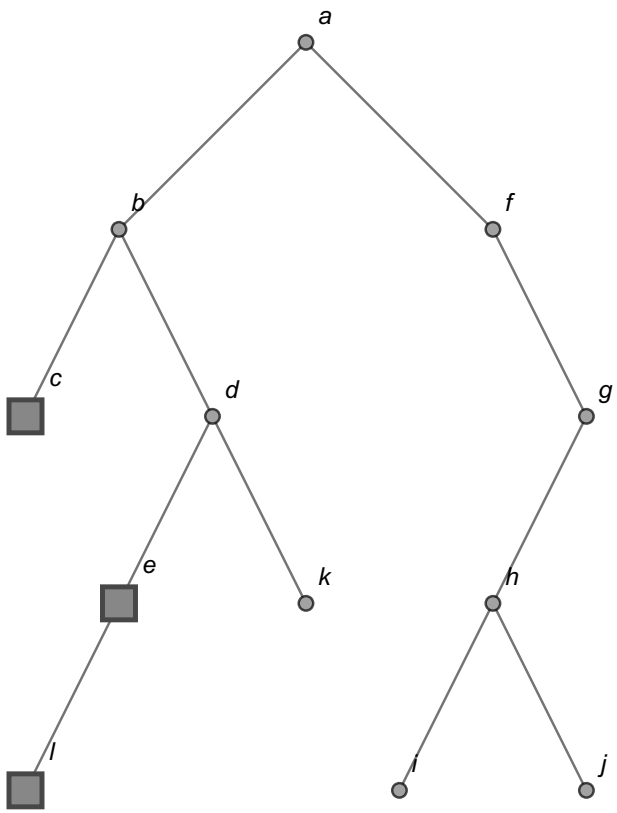
Solución del ejemplo 6.12

- Se regresa a b, se pasa a la derecha de b en d, a la izquierda de d en e, a la izquierda de e en l y se visita l :



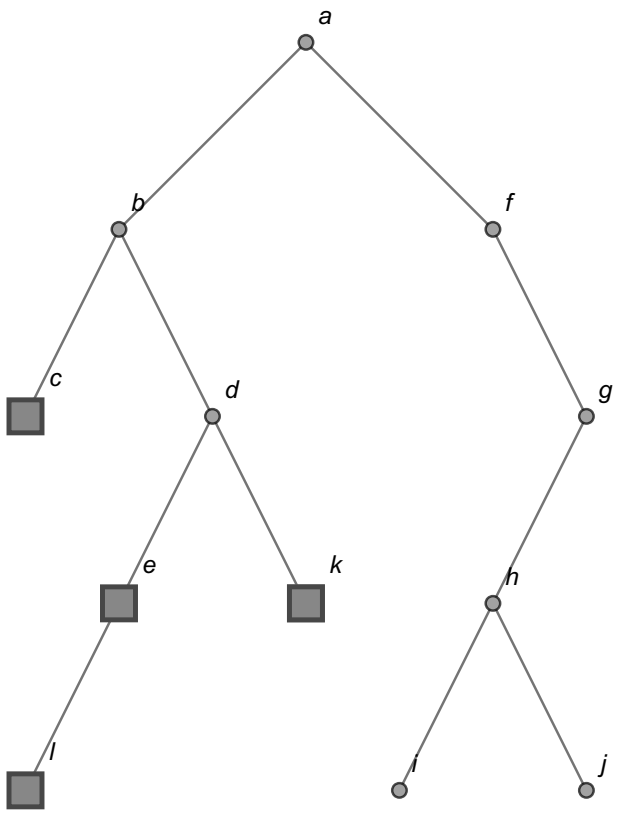
Solución del ejemplo 6.12

- Se regresa a e y como no hay nada a su derecha, se imprime e :



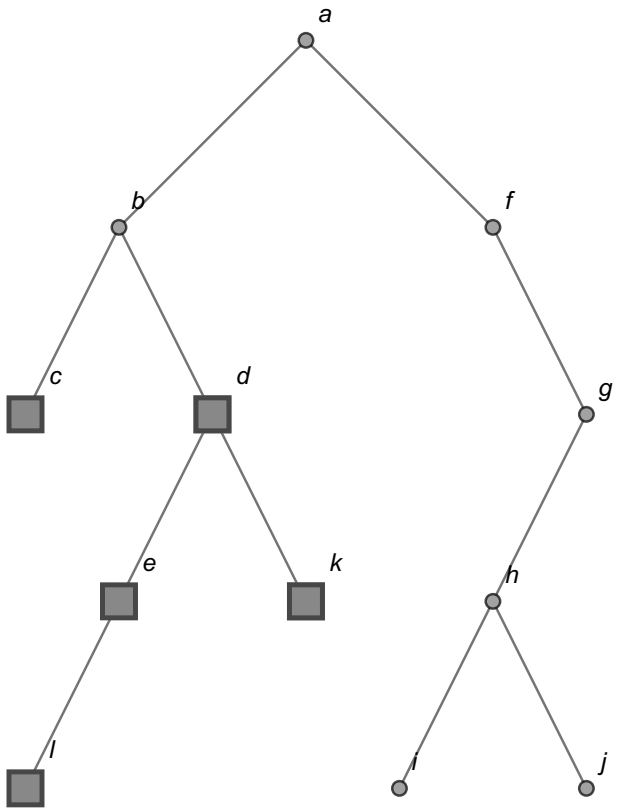
Solución del ejemplo 6.12

- Se regresa a d, se pasa a su derecha y se visita k :



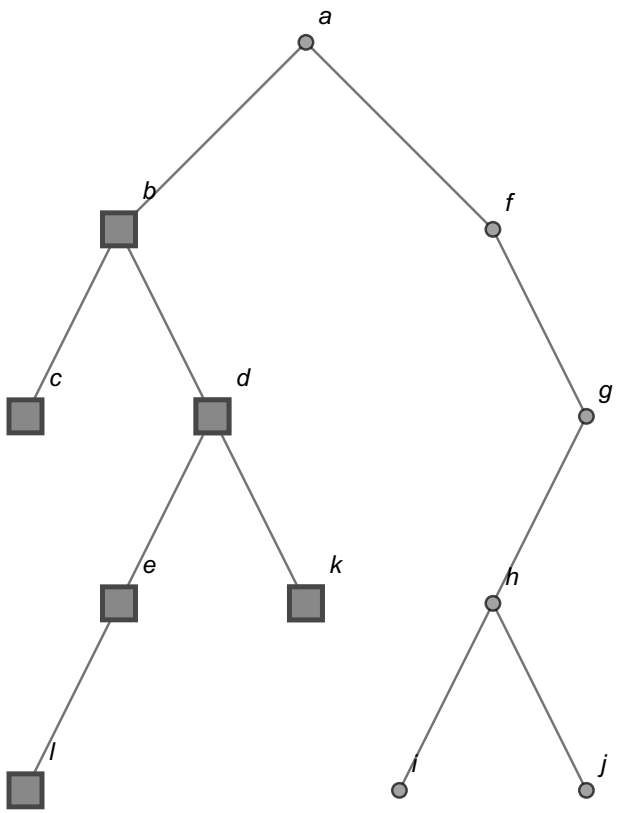
Solución del ejemplo 6.12

- Se regresa a d y se imprime d :



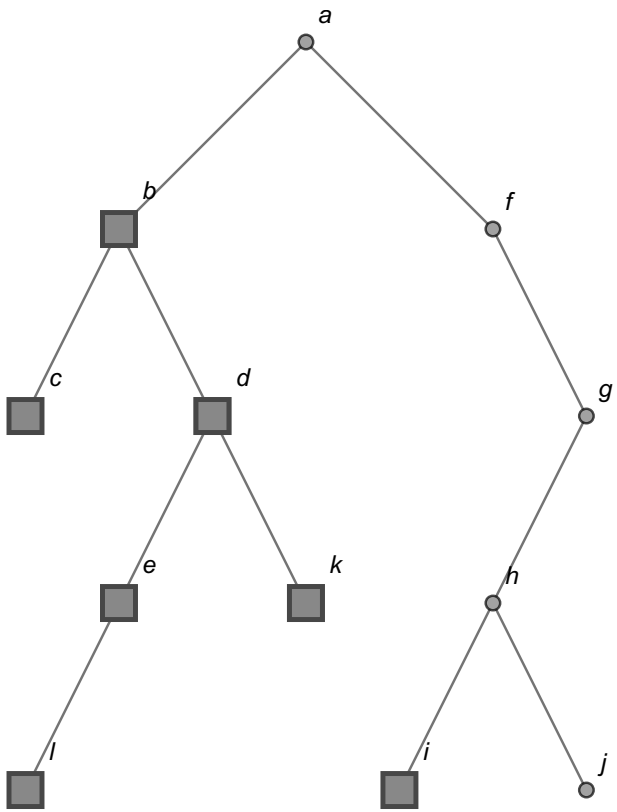
Solución del ejemplo 6.12

- Se regresa a b y se visita b :



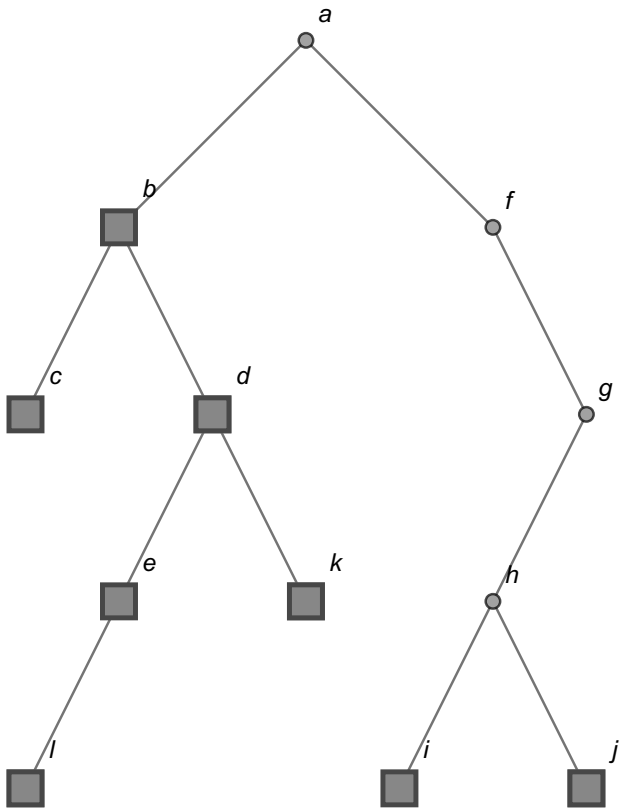
Solución del ejemplo 6.12

- Se regresa a a, se pasa a su derecha en f, en f como no hay nada a su izquierda se realiza un desplazamiento a su derecha en g, luego a la izquierda de g en h, a la izquierda de h en i y se imprime i :



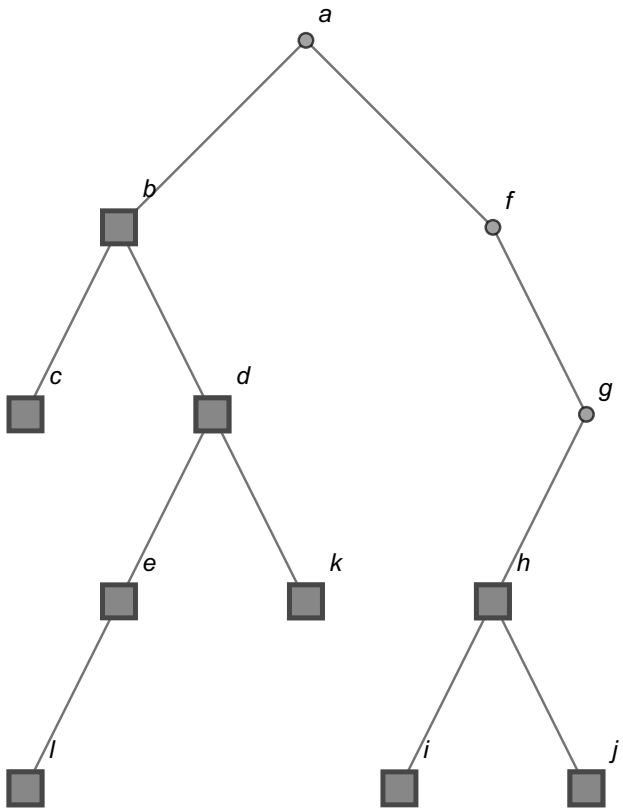
Solución del ejemplo 6.12

- Se regresa a h, se pasa a su derecha y se visita j :



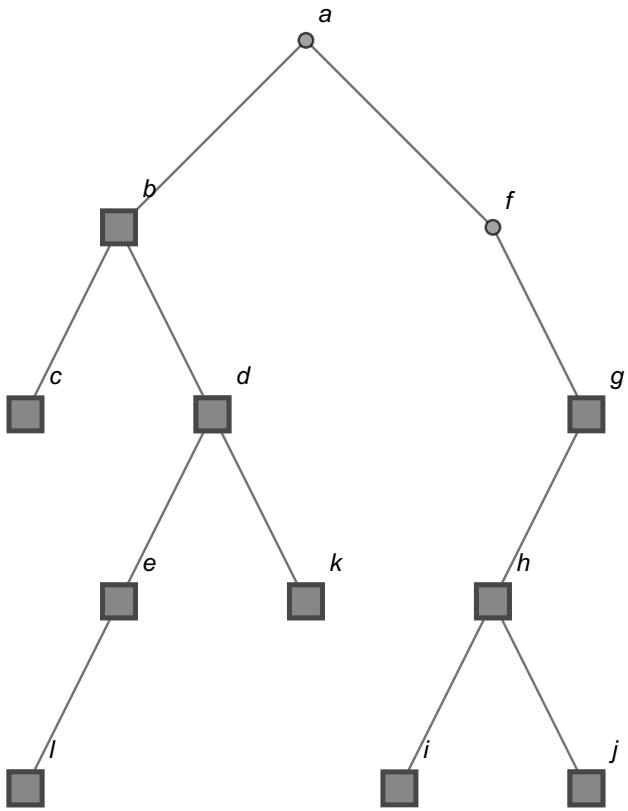
Solución del ejemplo 6.12

- Se regresa a h y se imprime h :



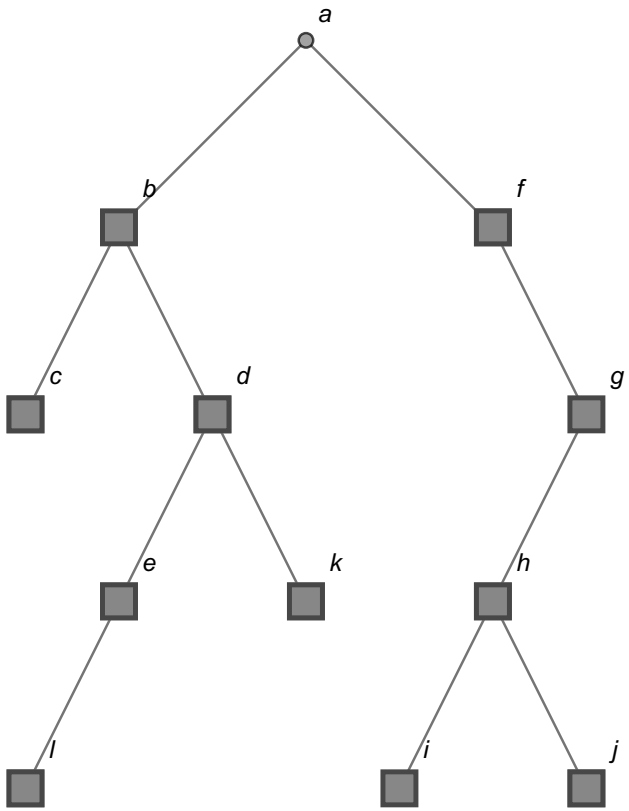
Solución del ejemplo 6.12

- Se regresa a g, no hay nada a su derecha, por lo que, se imprime g :



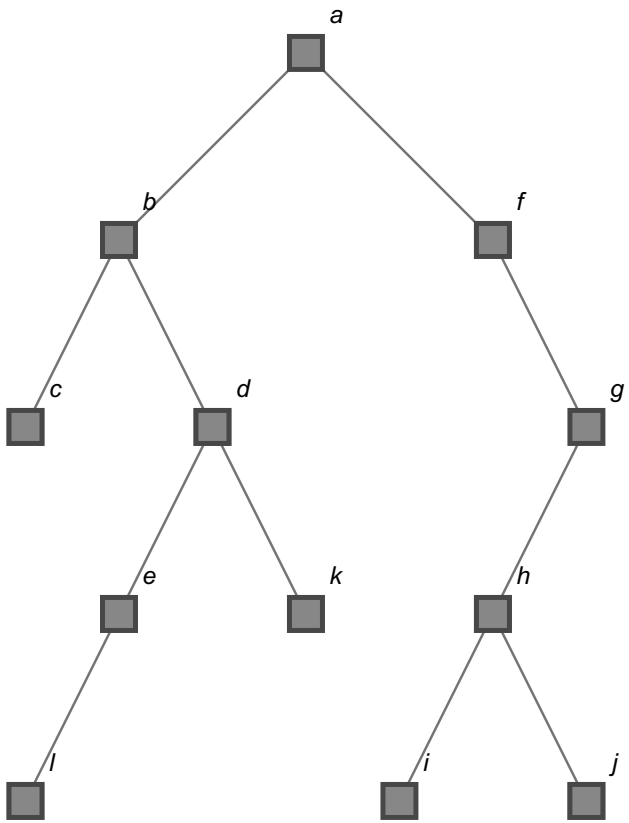
Solución del ejemplo 6.12

- Se regresa a f y se visita f :



Solución del ejemplo 6.12

- Se regresa a *a* y se imprime *a* :



Solución del ejemplo 6.12

- En resumen, el recorrido postorden es: $\{c, l, e, k, d, b, i, j, h, g, f, a\}$.

Nota

El estudiante debe observar que la raíz del árbol binario empleado en cada uno de los recorridos, queda en una posición correspondiente al orden utilizado. En el recorrido prefijo, la raíz *a* en este ejemplo, quedó al principio. En el recorrido interfijo, la raíz *a* ocupa más o menos una posición central o intermedia. En el recorrido postfijo, la raíz *a* se visitó hasta el final del proceso. En general, en cualquier árbol binario siempre se manifestará este comportamiento con relación a la raíz, al usar los algoritmos del teorema 16.

Solución del ejemplo 6.12

El software *Mathematica* cuenta con los comandos `Prefijo` y `Postfijo` para realizar los recorridos de orden inicial y orden final sobre un árbol binario pasado como argumento, especificando además, la raíz del árbol. Estas instrucciones forman parte del paquete **VilCretas** y ambas presentan la opción `animacion -> True` que despliega una animación para visualizar paso a paso, cada uno de los vértices visitados en el orden respectivo. Al emplear estas sentencias en el presente ejercicio, se tiene:

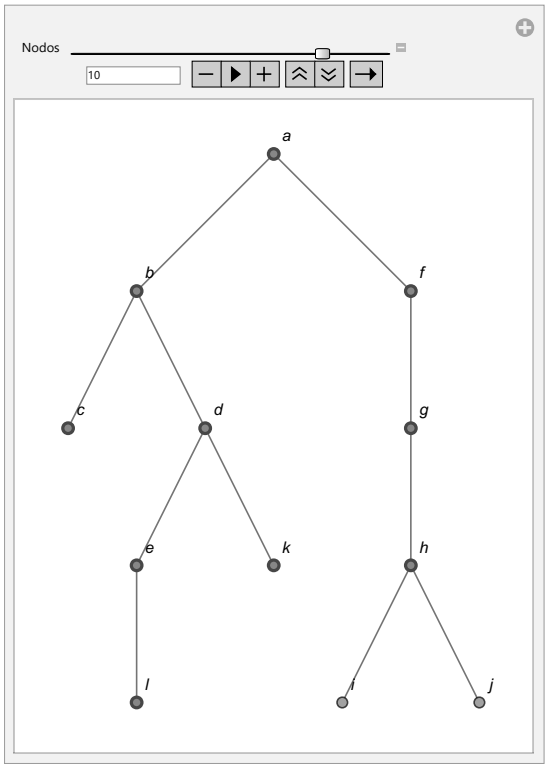
In[ ] :=

```
arbol = Arbol[{{a, b}, {a, f}, {b, c}, {b, d}, {f, g},
{d, e}, {d, k}, {g, h}, {h, i}, {h, j}, {e, l}}];
Prefijo[arbol, a]
Prefijo[arbol, a, animacion -> True]
Postfijo[arbol, a]
Postfijo[arbol, a, animacion -> True]
```

Solución del ejemplo 6.12

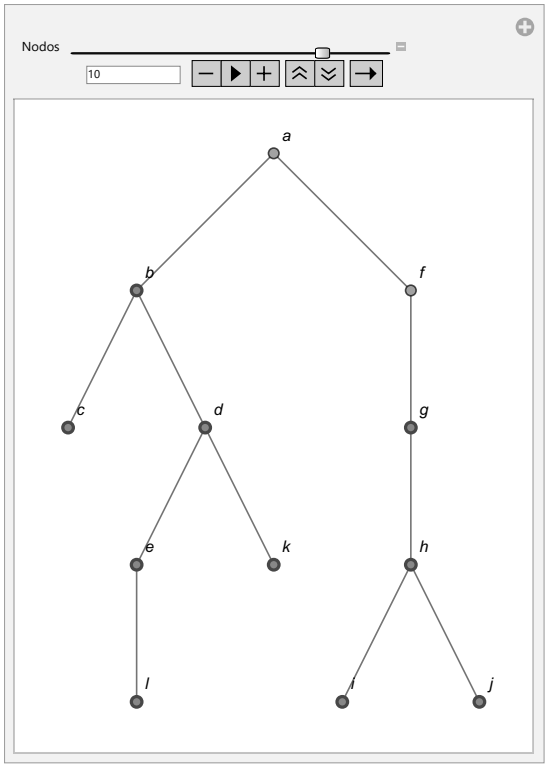
Out[ ] =

{a, b, c, d, e, l, k, f, g, h, i, j}
{a, b, c, d, e, l, k, f, g, h, i, j}



Solución del ejemplo 6.12

{c, l, e, k, d, b, i, j, h, g, f, a}
{c, l, e, k, d, b, i, j, h, g, f, a}



Solución del ejemplo 6.12

Las listas de visitación de los nodos aparecen duplicadas en el **Out[ ]**, pues `Prefijo` y `Postfijo` se están llamando dos veces en el **In[ ]**, una de ellas con el atributo `animacion -> True`. El árbol dibujado por *Wolfram* en las animaciones, presenta algunas aristas verticales en aquellos padres que solo tienen un hijo, esta limitación del software ya se habia señalado con anterioridad.

Nota

Se aclara al estudiante que en la librería **VilCretas** no existe ningún comando que ejecute el recorrido de orden intermedio. También es esencial señalar, que las sentencias `Prefijo` y `Postfijo` pueden recibir como parámetro un árbol no necesariamente binario. Como ya se mencionó en la página 171, los recorridos prefijo y postfijo son generalizables a árboles de orden k, $k \in \mathbb{N}$ y las instrucciones `Prefijo` y `Postfijo` contemplan esa posibilidad.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-148.zip

Uso de la instrucción `Prefijo`

Explicación en video

https://youtu.be/YTFGNeS9AOA

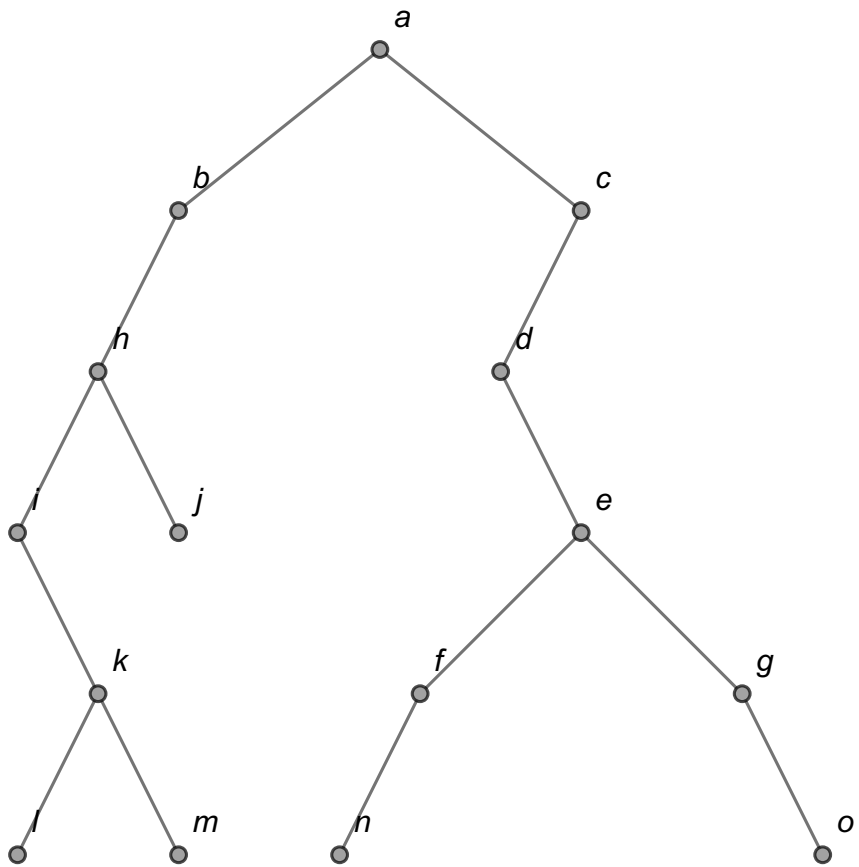
Utilización de la sentencia `Postfijo`

Explicación en video

https://youtu.be/37EJvuMI-XE

Example (6.13)

Realice los recorridos preorden, interorden y postorden en el árbol:

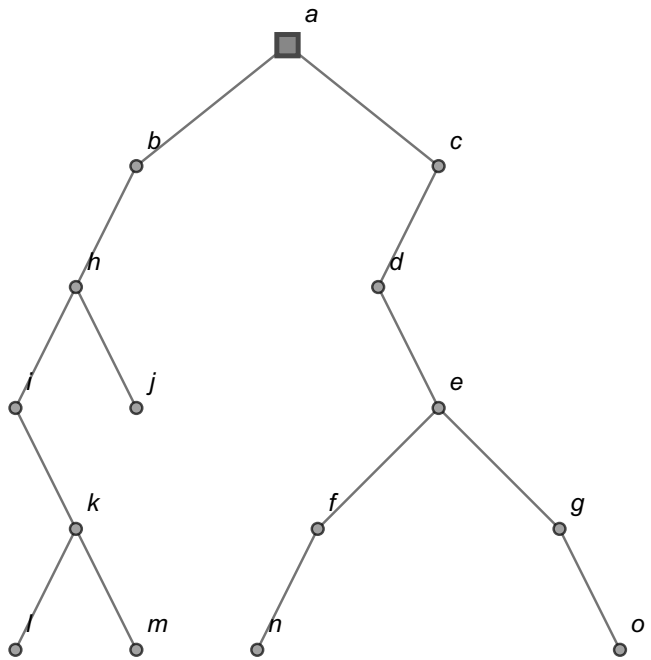


Solución del ejemplo 6.13

De forma equivalente al ejemplo 18, cada recorrido será puntualizado según lo propuesto por el teorema 16.

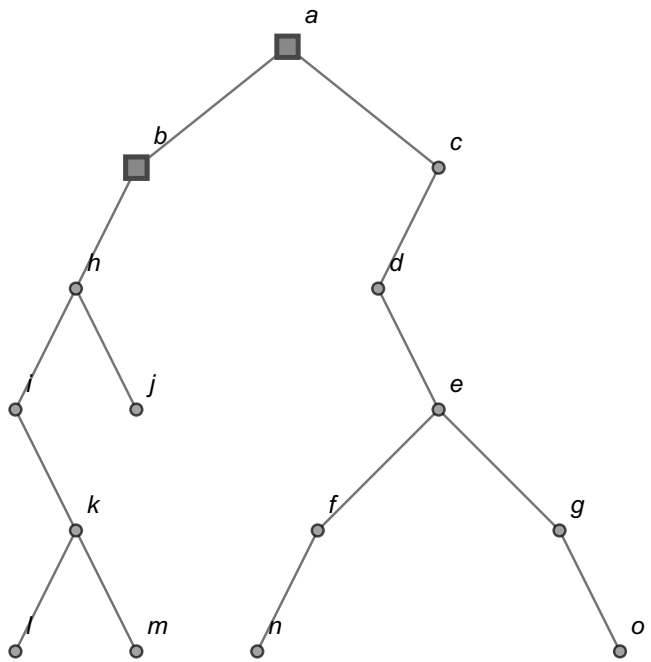
Recorrido preorden:

- Se imprime la raíz a :



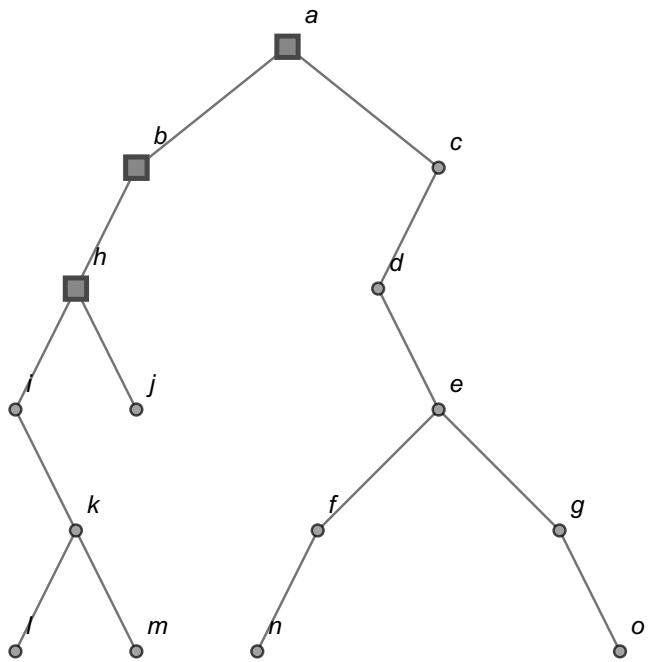
Solución del ejemplo 6.13

- Se pasa a la izquierda de a y se visita b :



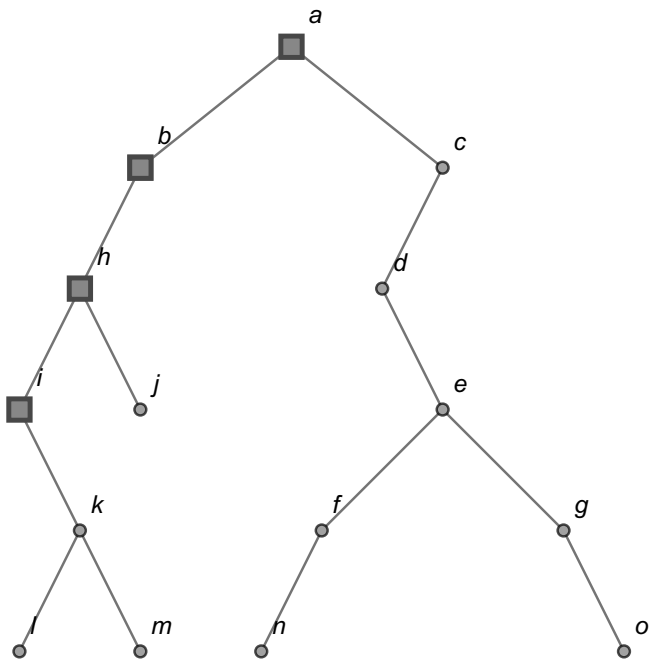
Solución del ejemplo 6.13

- Se realiza un desplazamiento a la izquierda de b y se imprime h :



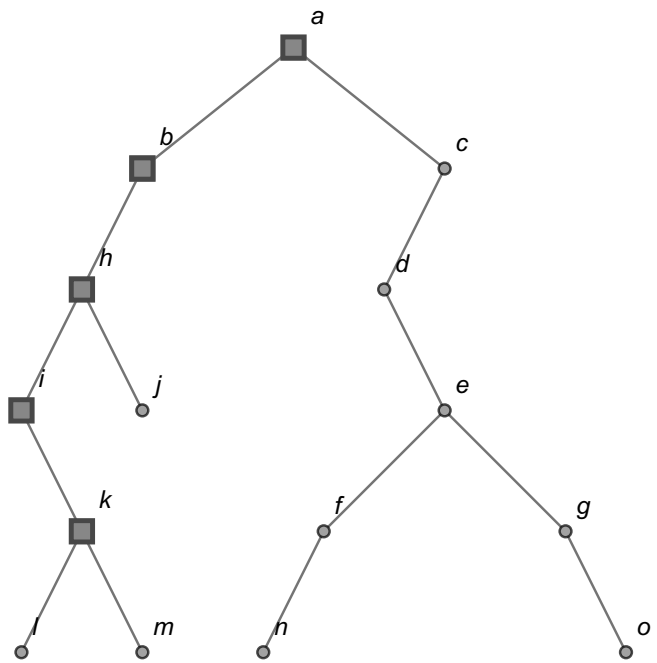
Solución del ejemplo 6.13

- Se ejecuta un traslado a la izquierda de h y se visita el nodo i :



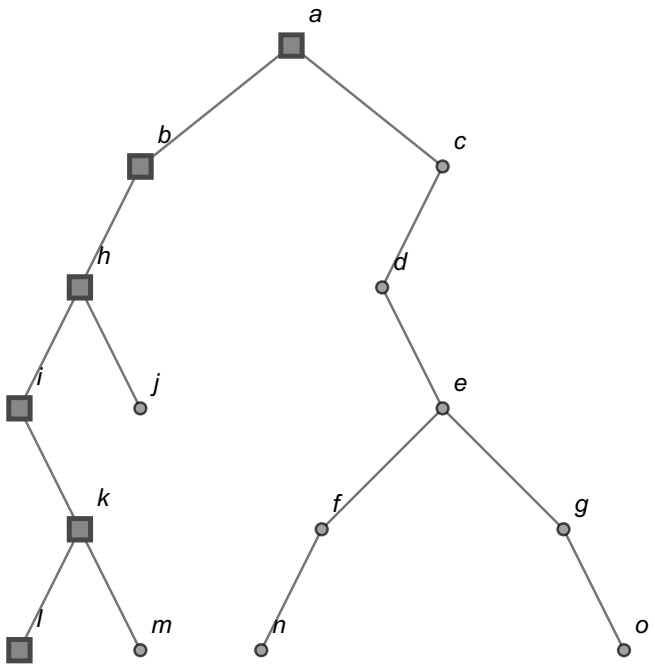
Solución del ejemplo 6.13

- No hay nada a la izquierda de i, por lo que, se toma la dirección derecha y se imprime k :



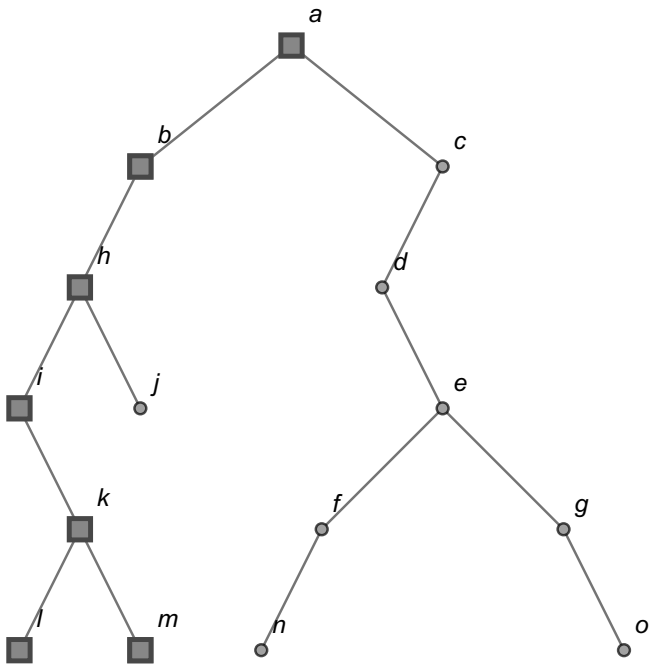
Solución del ejemplo 6.13

- Se pasa a la izquierda de k y se visita l :



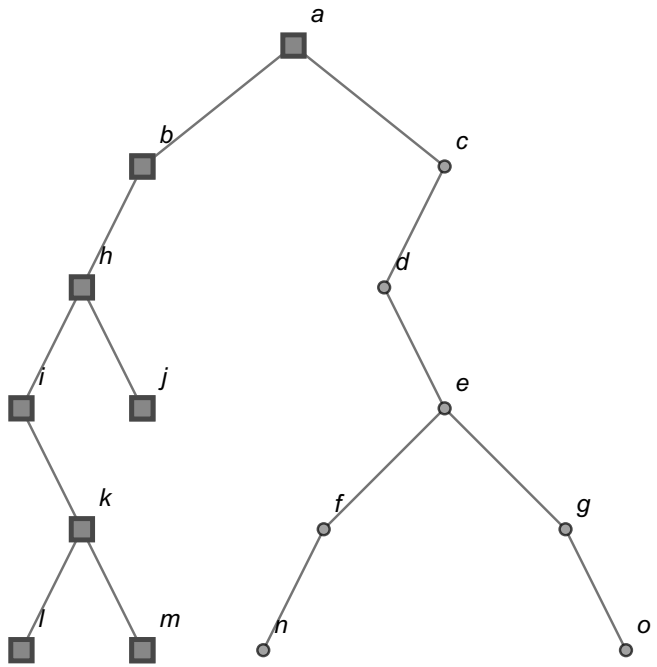
Solución del ejemplo 6.13

- Se regresa a k, se toma su derecha y se imprime m :



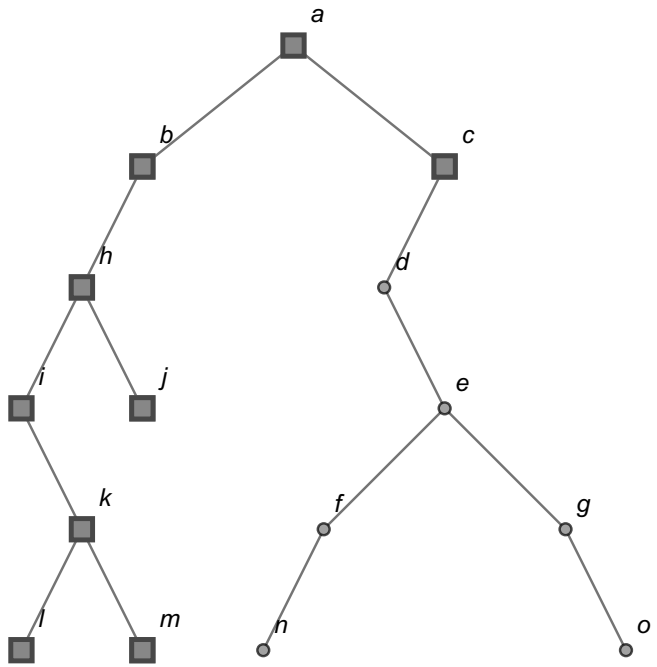
Solución del ejemplo 6.13

- Se regresa a k, i y h, se pasa a la derecha de h y se imprime j :



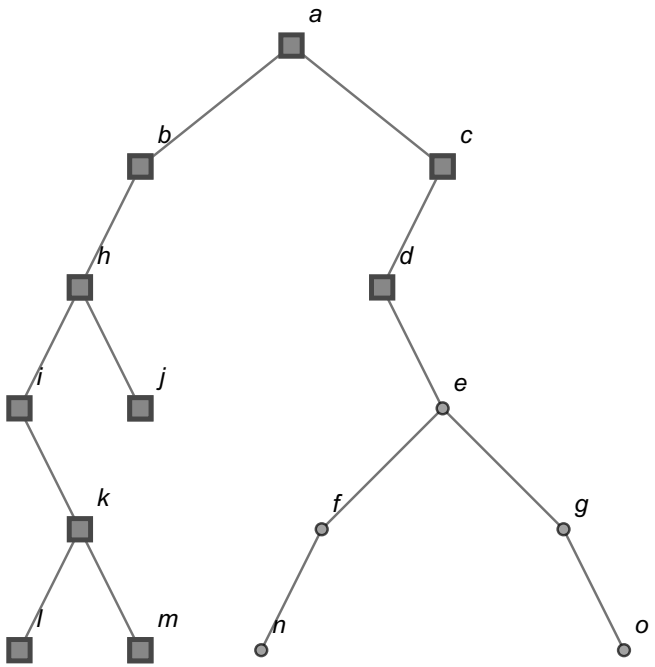
Solución del ejemplo 6.13

- Se regresa a h, b y a, se toma la derecha de a y se visita el vértice c :



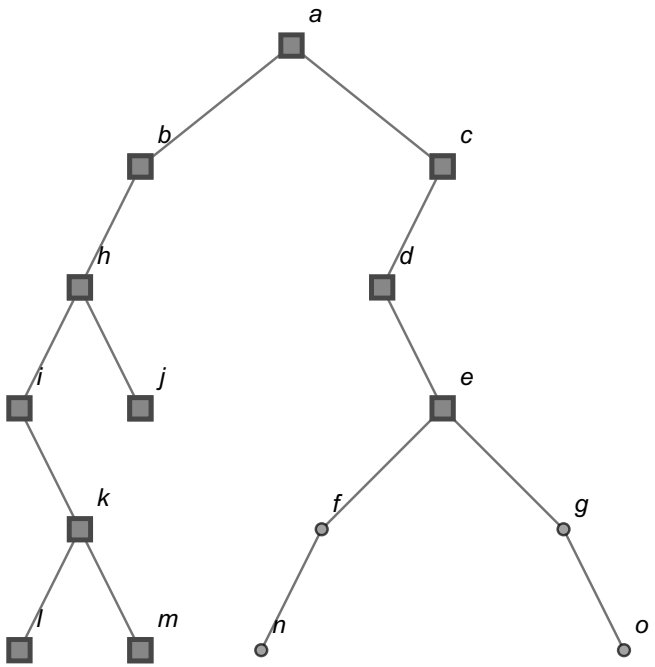
Solución del ejemplo 6.13

- Se pasa a la izquierda de c y se imprime d :



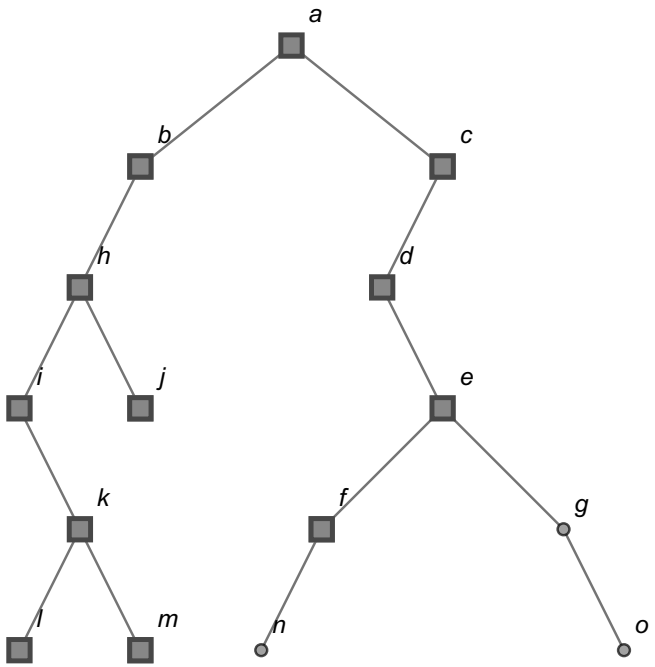
Solución del ejemplo 6.13

- No hay nada a la izquierda de d, se toma su derecha y se imprime e :



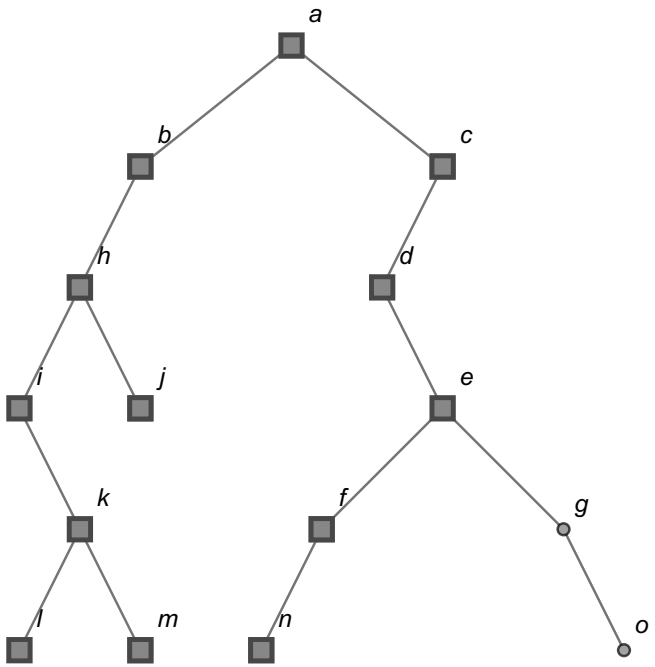
Solución del ejemplo 6.13

- Se pasa a la izquierda de e y se imprime f :



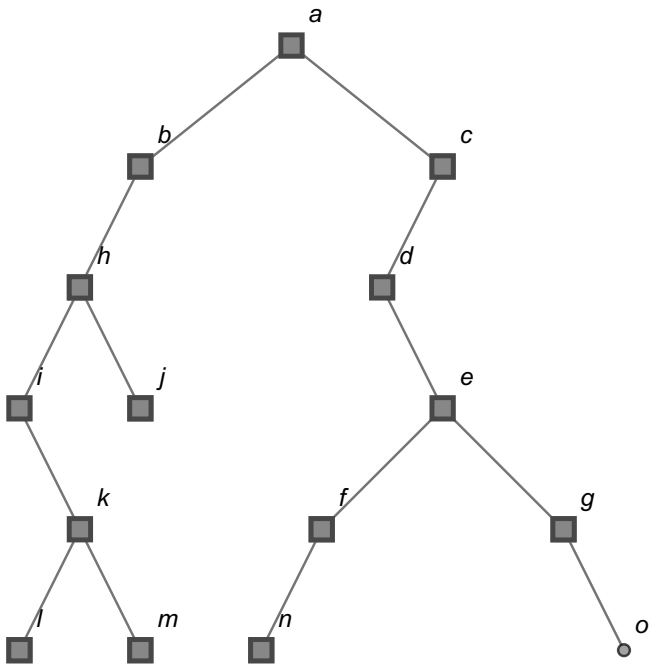
Solución del ejemplo 6.13

- Se pasa a la izquierda de f y se visita n :



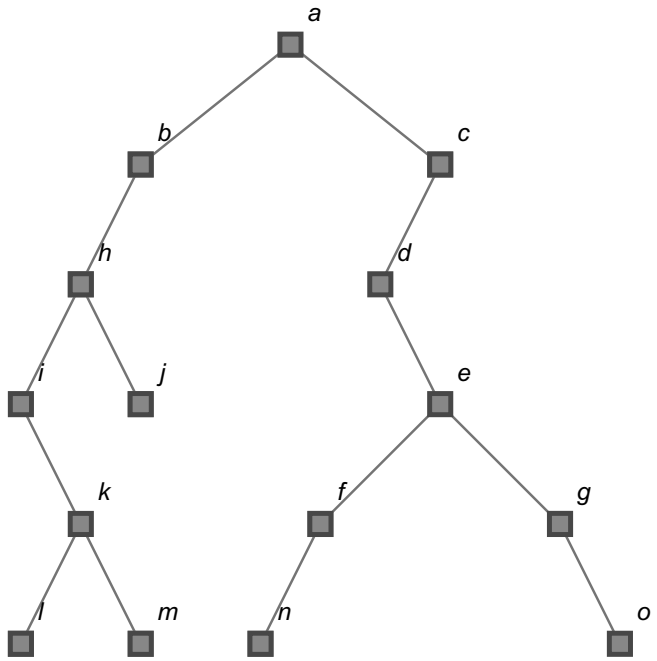
Solución del ejemplo 6.13

- Se regresa a f y e, se pasa a la derecha de e y se visita g :



Solución del ejemplo 6.13

- Como no hay nada a la izquierda de g, se toma su derecha y se imprime o :



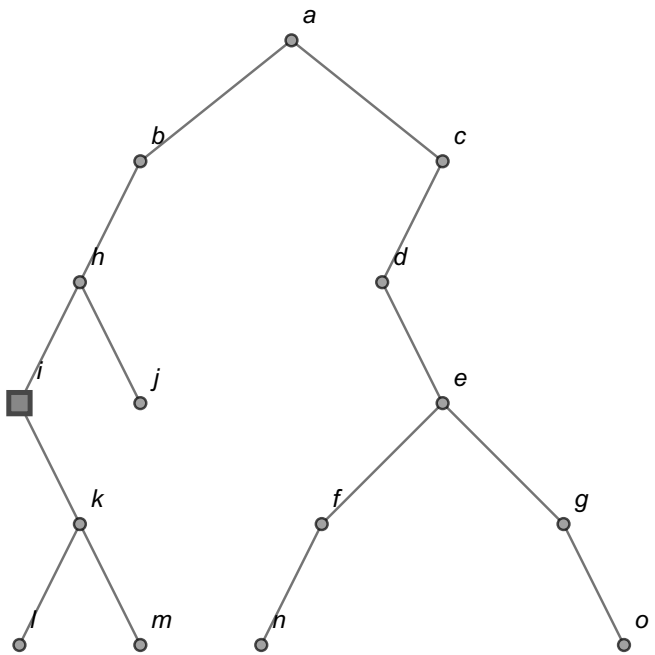
Solución del ejemplo 6.13

- Se regresa a g, e, d, c y a. Finalmente, el recorrido prefijo corresponde a: $\{a, b, h, i, k, l, m, j, c, d, e, f, n, g, o\}$.

Solución del ejemplo 6.13

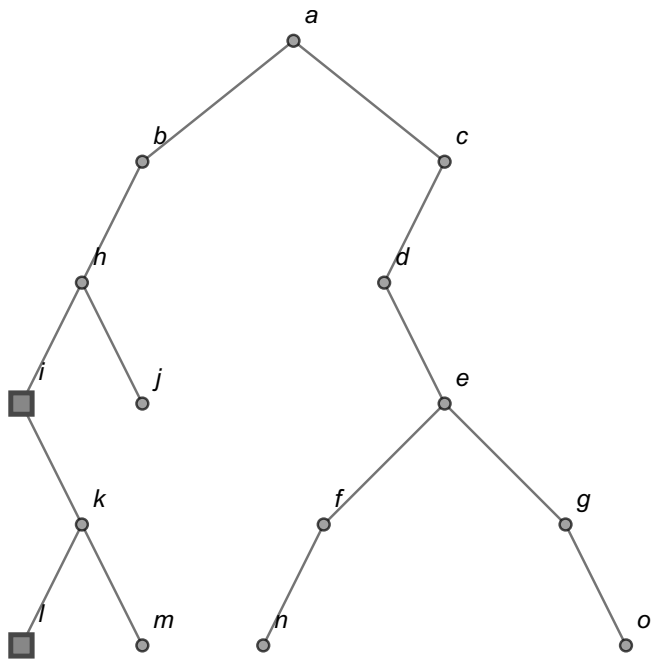
Recorrido interorden:

- Se comienza en la raíz a, se toma su izquierda en b, se pasa a la izquierda de b en h, se toma su izquierda en i y se imprime i :



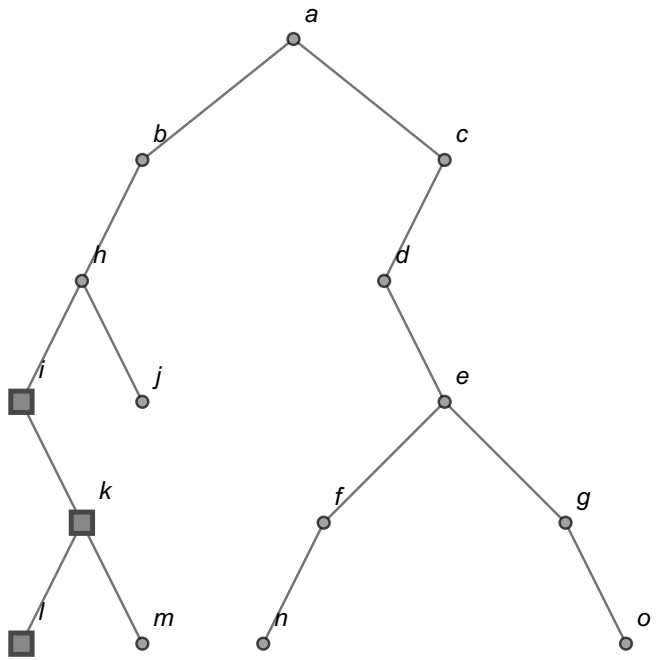
Solución del ejemplo 6.13

- Se pasa a la derecha de i en k, luego a la izquierda de k en l y se visita l :



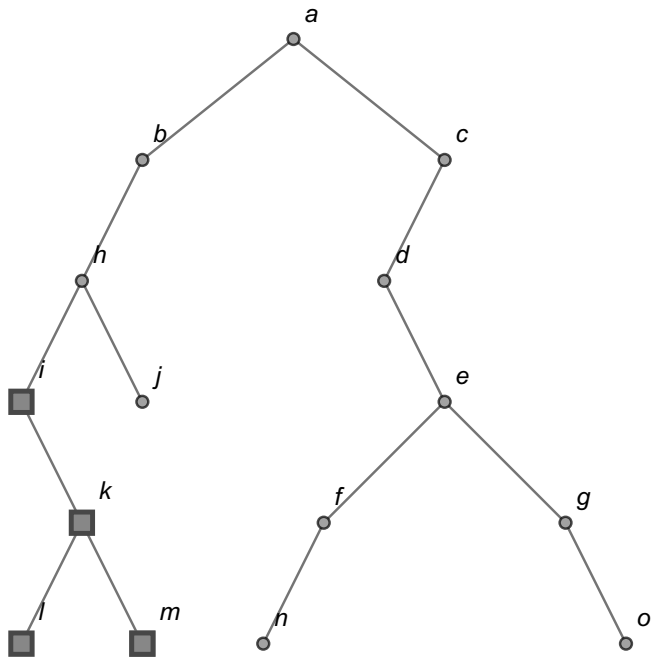
Solución del ejemplo 6.13

- Se regresa a k y se imprime k :



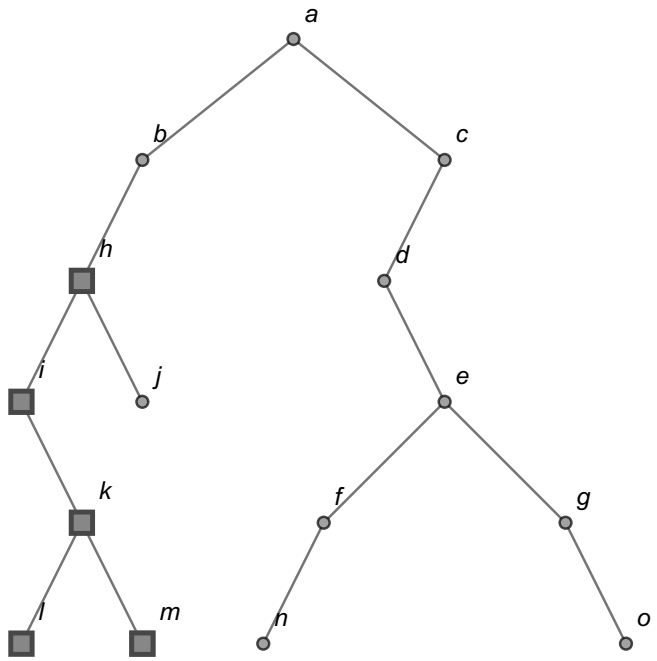
Solución del ejemplo 6.13

- Se pasa a la derecha de k y se visita m :



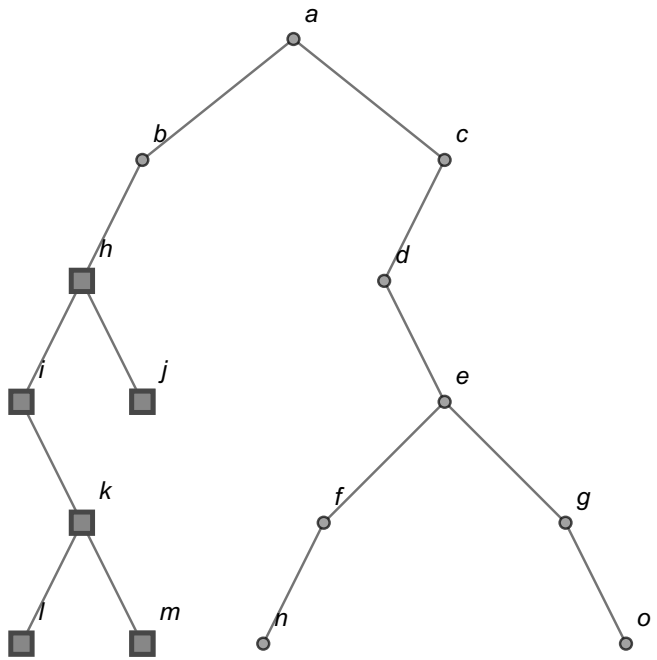
Solución del ejemplo 6.13

- Se regresa a k, i y h, se imprime h :



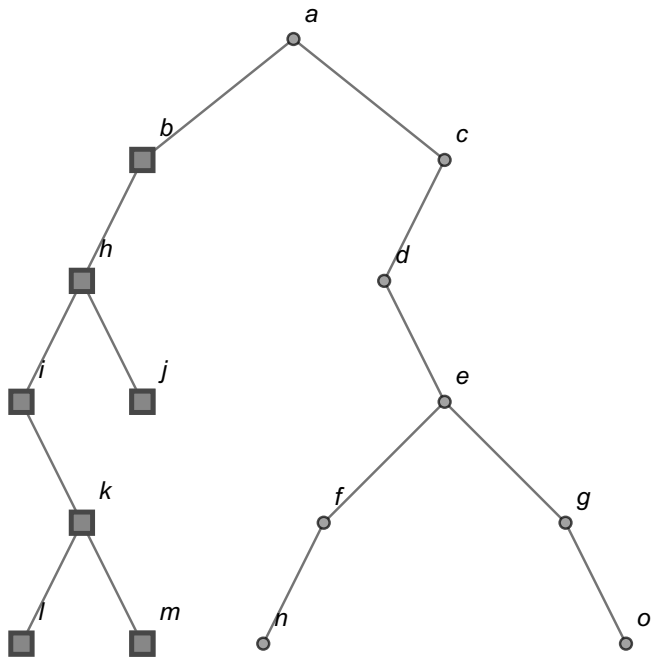
Solución del ejemplo 6.13

- Se toma la derecha de h y se visita j :



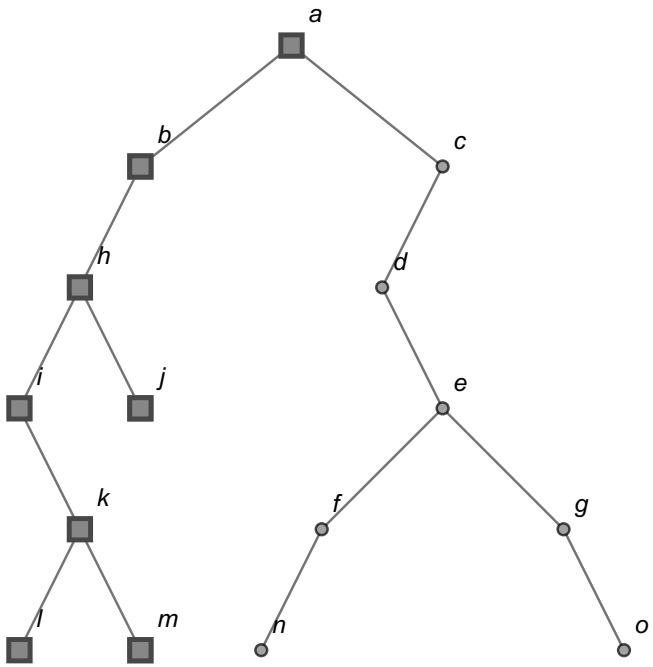
Solución del ejemplo 6.13

- Se regresa a h y b, se imprime b :



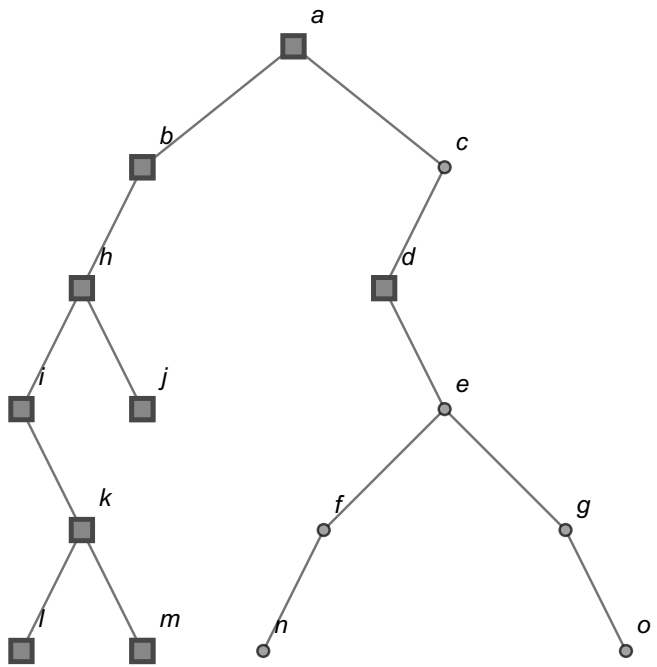
Solución del ejemplo 6.13

- Como no hay nodos a la derecha de b, se regresa a a y se imprime a :



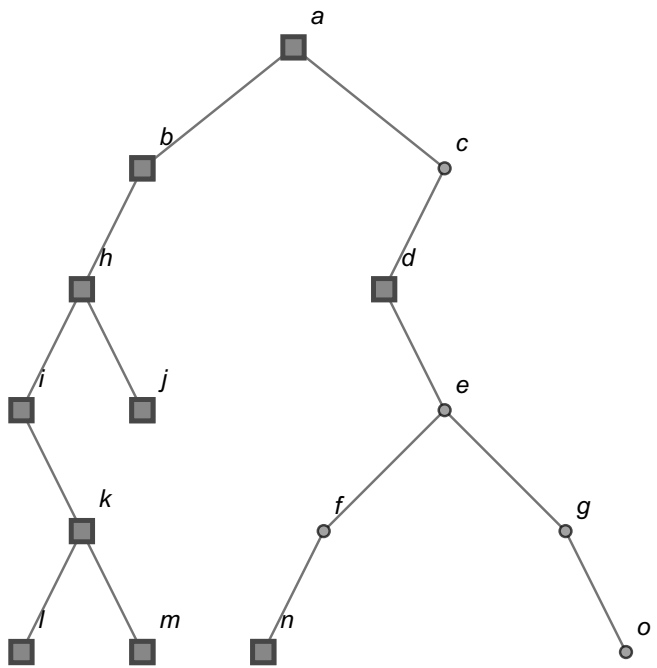
Solución del ejemplo 6.13

- Se pasa a la derecha de a en c, se toma la izquierda de c en d y se visita d :



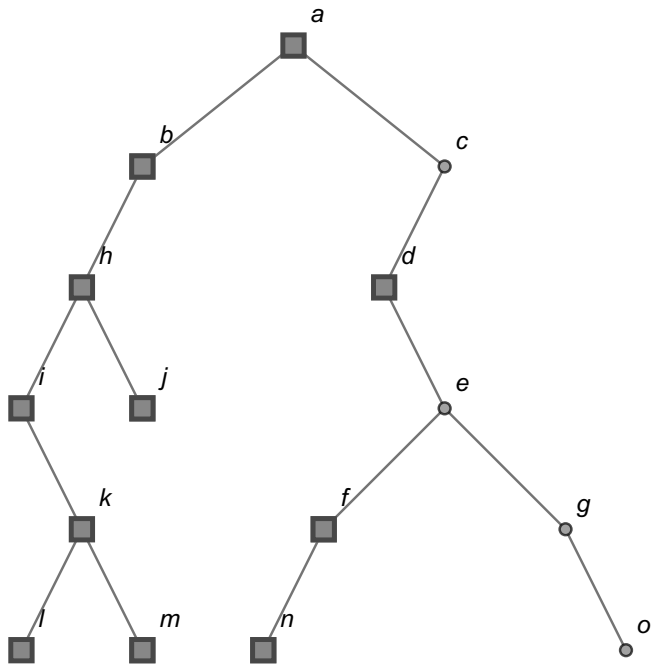
Solución del ejemplo 6.13

- Se toma la derecha de d en e, se pasa a su izquierda en f y a la izquierda de f en n, se visita n :



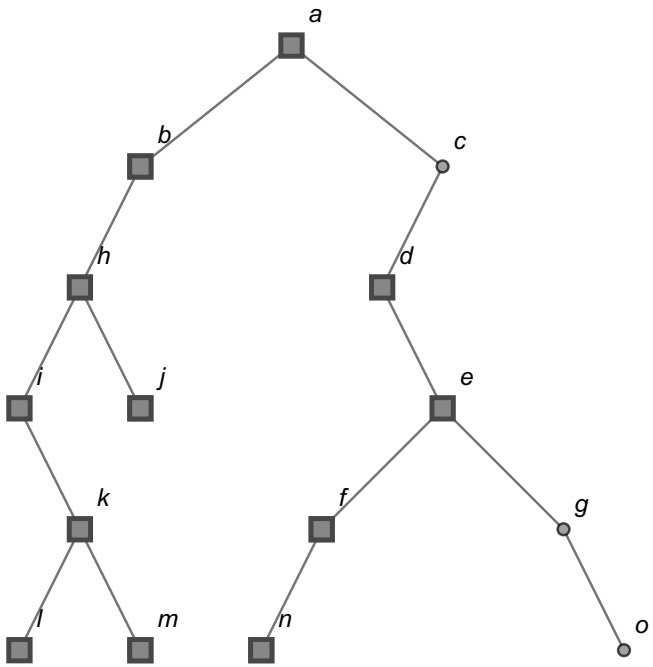
Solución del ejemplo 6.13

- Se regresa a f y se imprime f :



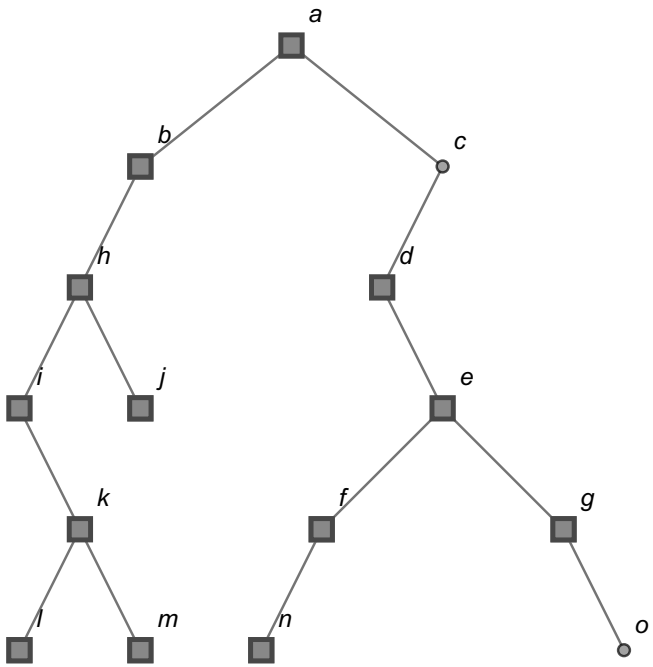
Solución del ejemplo 6.13

- Se regresa a e y se visita e :



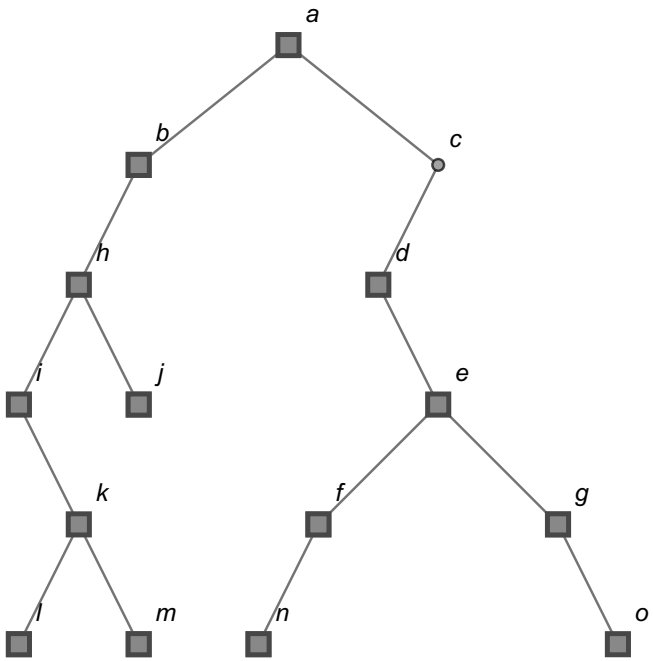
Solución del ejemplo 6.13

- Se toma la derecha de e y se imprime g :



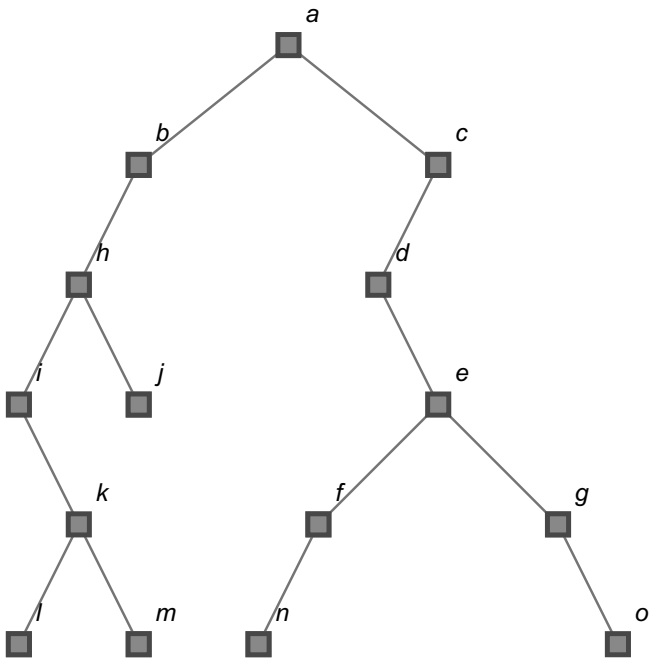
Solución del ejemplo 6.13

- Se pasa a la derecha de g y se visita o :



Solución del ejemplo 6.13

- Se regresa a g, e, d y c, se imprime c :



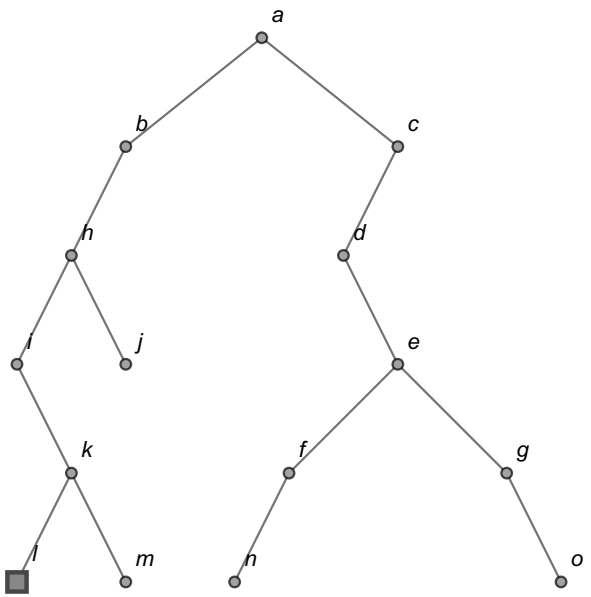
Solución del ejemplo 6.13

- Finalmente, se regresa al vértice a. El recorrido interfijo es: $\{i, l, k, m, h, j, b, a, d, n, f, e, g, o, c\}$.

Solución del ejemplo 6.13

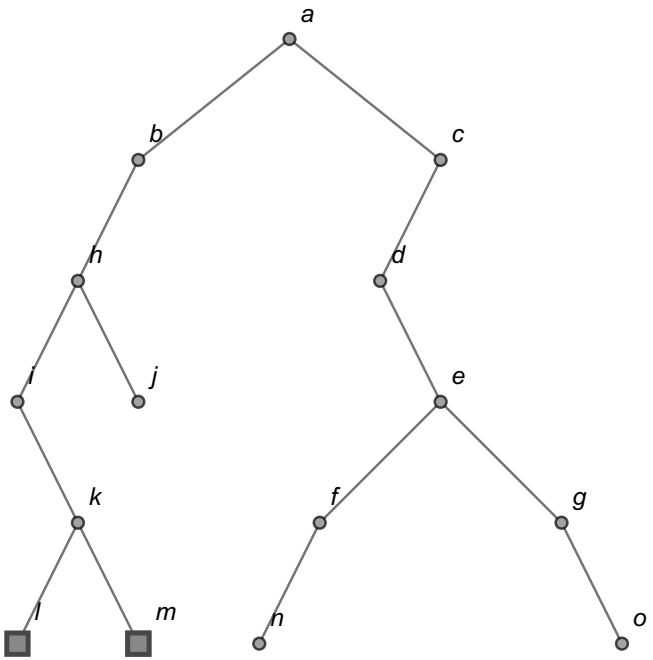
Recorrido postorden:

- Ubicados en la raíz a del árbol, se toma su izquierda en b, la izquierda de b en h, la izquierda de h en i, al no haber nada a la izquierda de i, se realiza un desplazamiento a la derecha de i en k, luego a la izquierda de k en l y se imprime l :



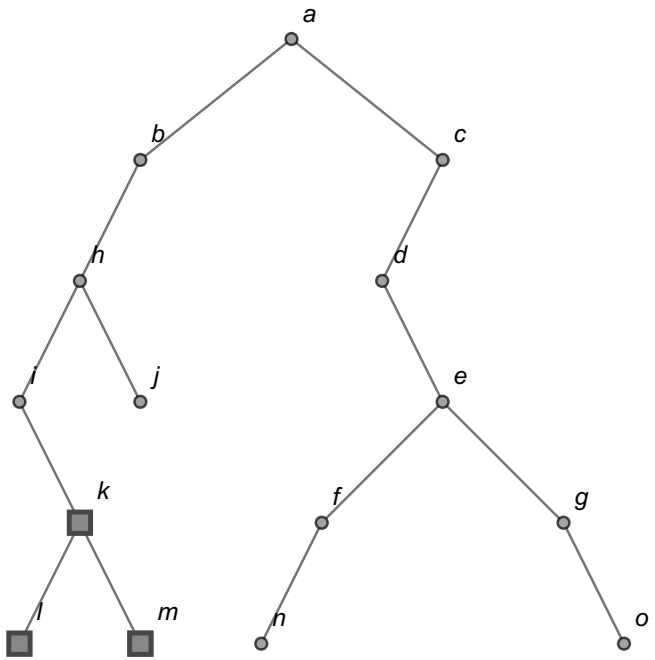
Solución del ejemplo 6.13

- Se regresa a k, se toma su derecha en m y se visita m :



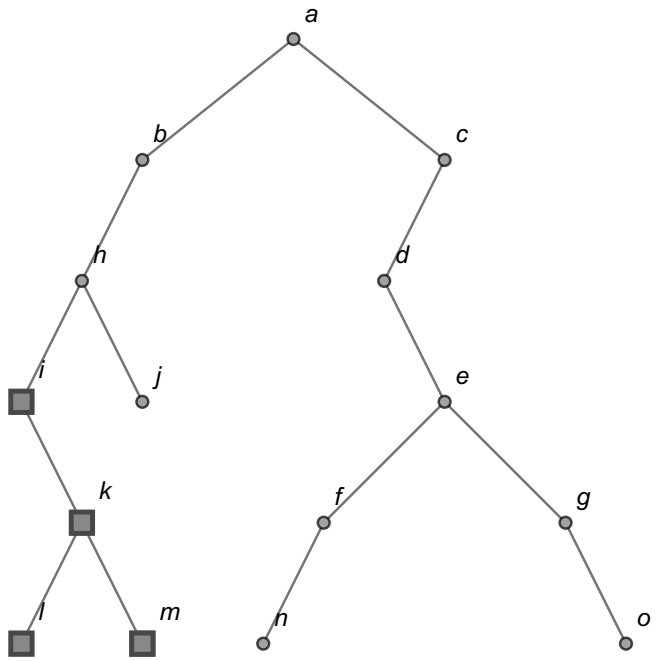
Solución del ejemplo 6.13

- Se regresa a k y se imprime k :



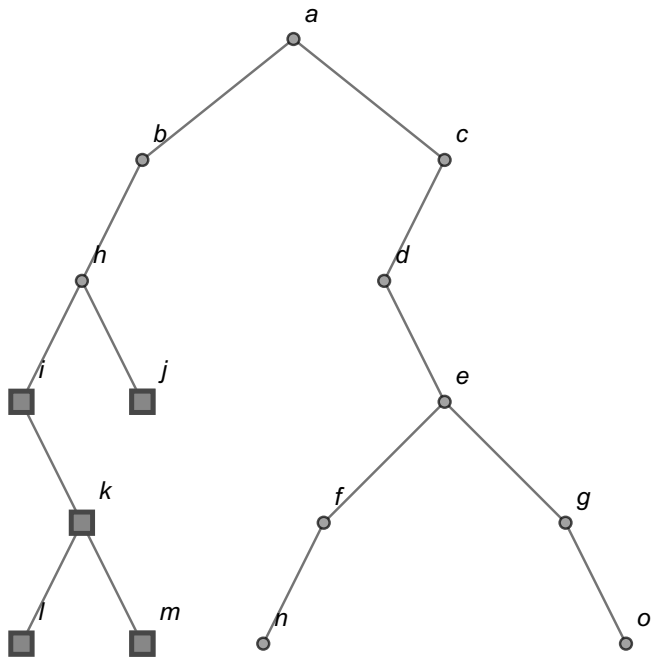
Solución del ejemplo 6.13

- Se regresa a i y se visita i :



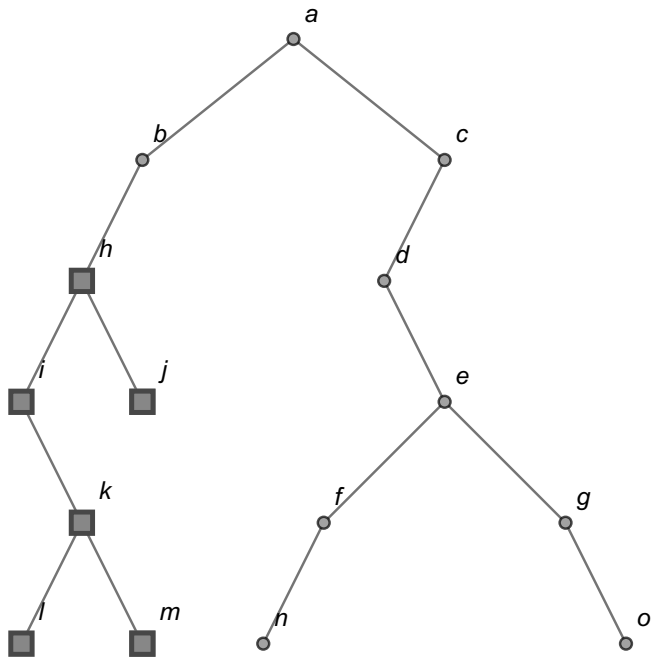
Solución del ejemplo 6.13

- Se regresa a h, se toma su derecha y se imprime j :



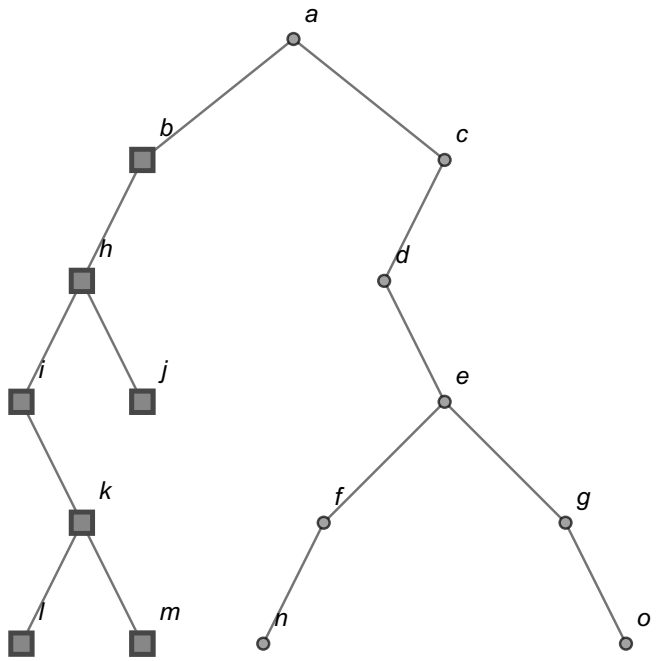
Solución del ejemplo 6.13

- Se regresa a h y se imprime h :



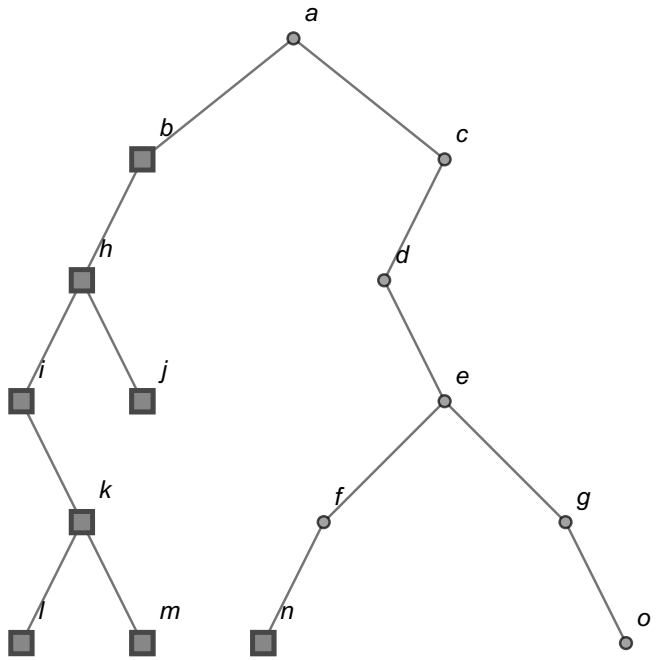
Solución del ejemplo 6.13

- Se regresa a b y se visita b :



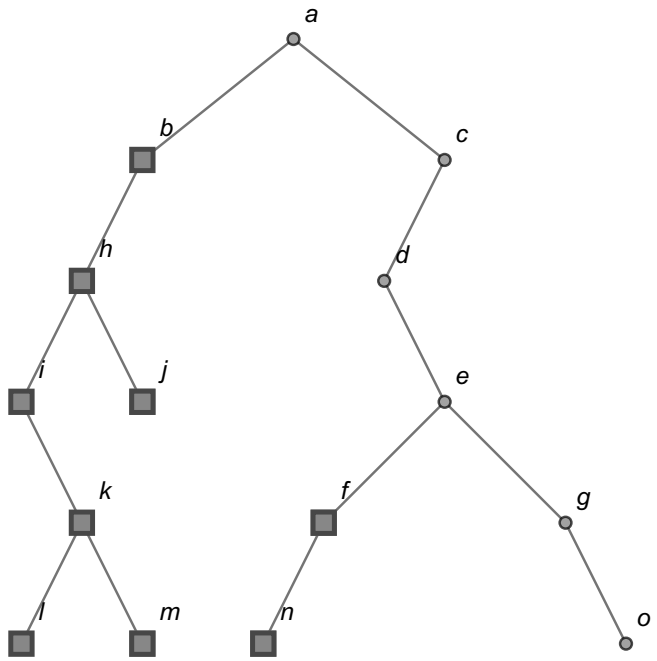
Solución del ejemplo 6.13

- Se regresa a a, se pasa a su derecha en c, a la izquierda de c en d, a la derecha de d en e, a su izquierda en f, a la izquierda de f en n y se imprime n :



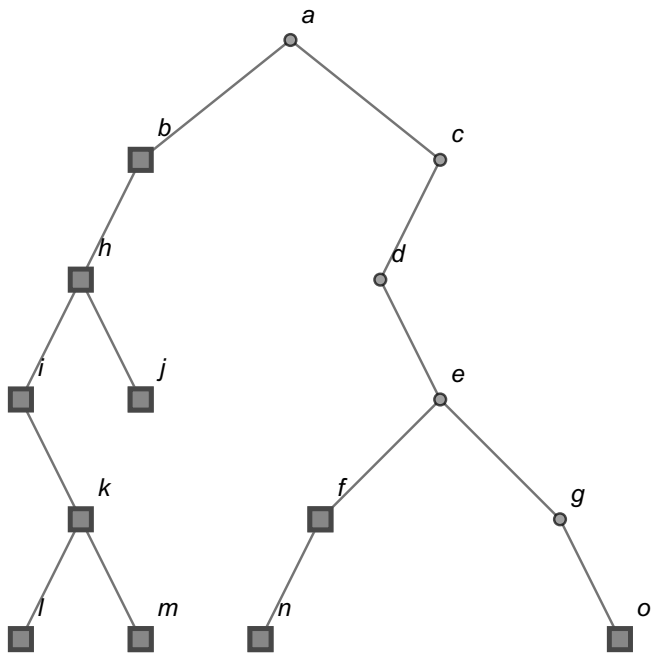
Solución del ejemplo 6.13

- Se regresa a f y se imprime f :



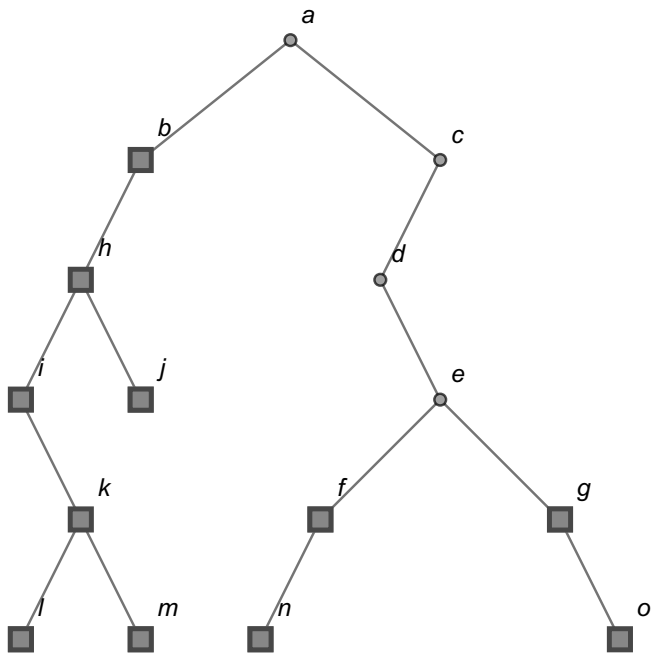
Solución del ejemplo 6.13

- Se regresa a e, se toma su derecha en g, al no haber nada a la izquierda de g, se pasa a su derecha y se visita o :



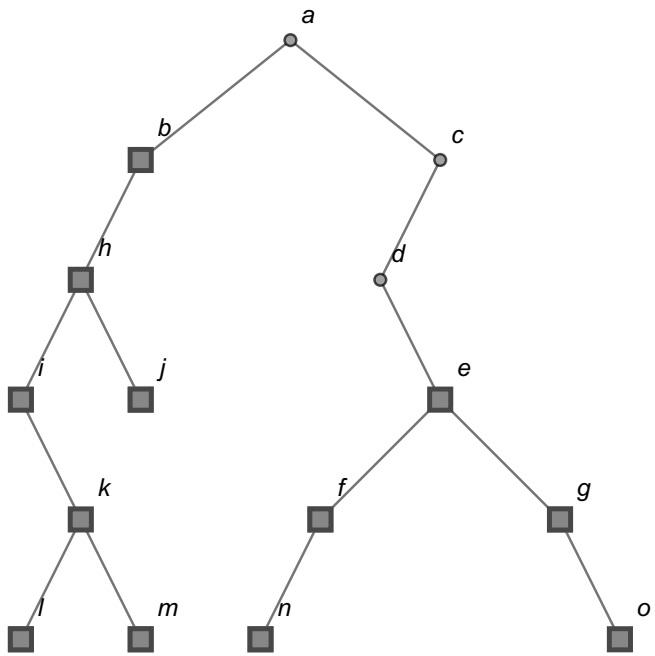
Solución del ejemplo 6.13

- Se regresa a g y se imprime g :



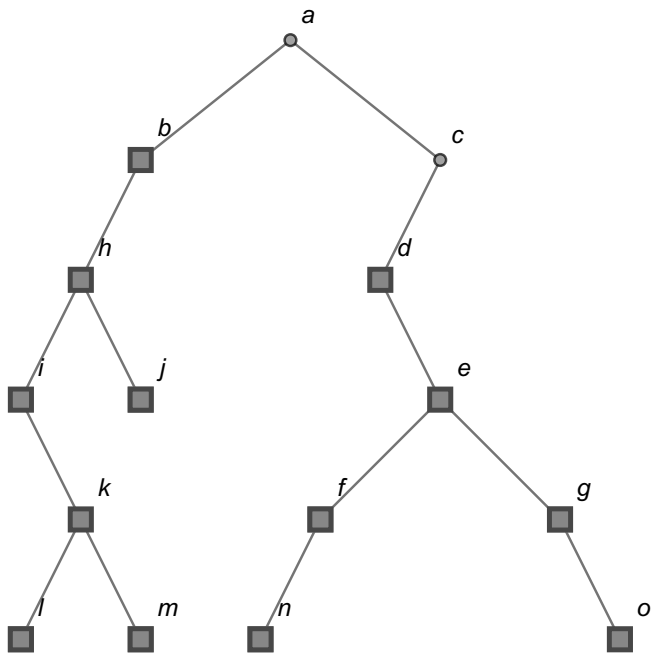
Solución del ejemplo 6.13

- Se regresa a e y se visita e :



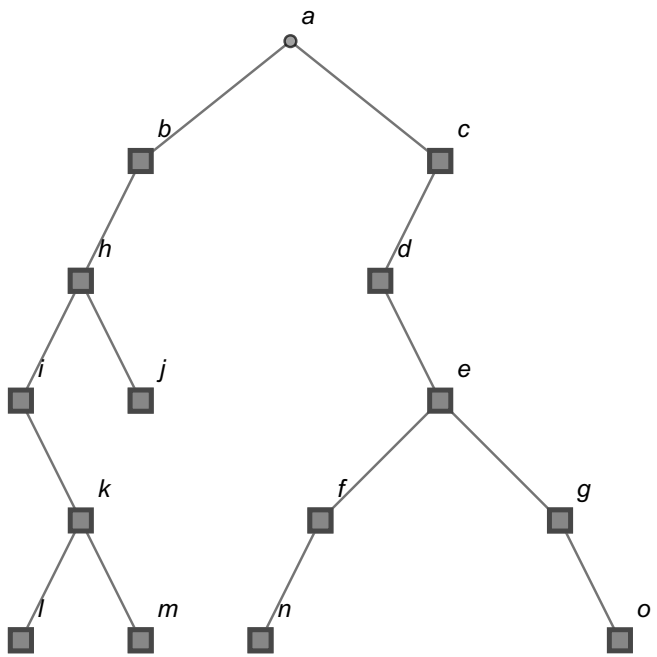
Solución del ejemplo 6.13

- Se regresa a d y se imprime d :



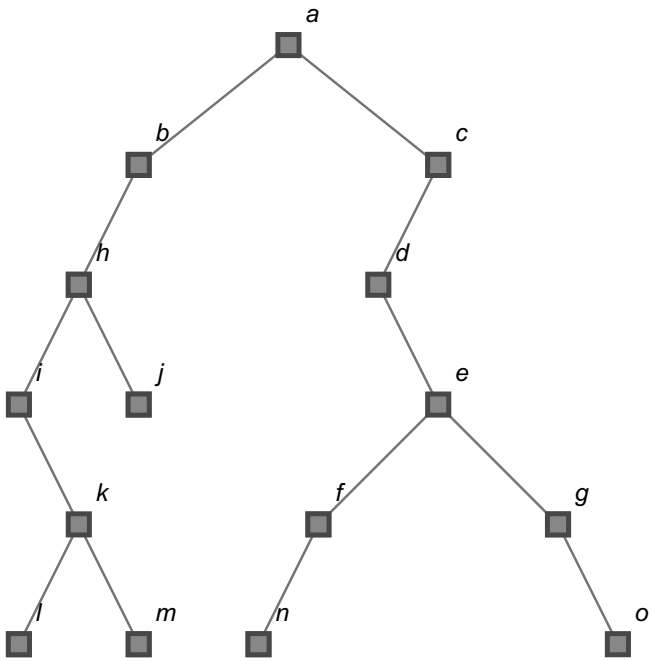
Solución del ejemplo 6.13

- Se regresa a c, al no existir vértices a la derecha, se imprime c :



Solución del ejemplo 6.13

- Se regresa a a y se visita a :



Solución del ejemplo 6.13

- El recorrido postfijo corresponde a: $\{l, m, k, i, j, h, b, n, f, o, g, e, d, c, a\}$.

Solución del ejemplo 6.13

Se insta al alumno a comprobar los recorridos de orden inicial y de orden final anteriores, usando las instrucciones `Prefijo` y `Postfijo` del paquete **VilCretas**.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-149.zip

Definición 6.2

Los recorridos inicial y final sobre un árbol binario completo proveniente de una expresión algebraica, proporcionan dos tipos de notaciones para la expresión: la "notación polaca" y la "notación polaca inversa". Consideremos la siguiente definición.

Definition (6.2)

Sea (T, r) un árbol binario completo de una expresión algebraica A y w una hilera formada por cada uno de los vértices de T entonces:

1. Si w es el resultado del recorrido prefijo sobre T, se llama a w notación polaca de A.
2. Si w se elaboró a través de un recorrido postfijo sobre T, se llama a w notación polaca inversa de A.

Comentario sobre la definición 20

En la definición 20 no se asocia ninguna notación con el recorrido interfijo de un árbol binario que representa una expresión algebraica, pues esta forma de visitación da como resultado la expresión original sin ningún paréntesis.
Por otro lado, de manera convencional desde la educación secundaria, el alumno está acostumbrado a utilizar la notación interfija de una expresión algebraica, sin embargo, esta presenta la dificultad de requerir el uso de paréntesis cuando existen de forma anidada, una combinación de operaciones aritméticas. En cambio, las notaciones polaca y polaca inversa, tienen la ventaja desde un punto de vista computacional, de no necesitar ningún tipo de paréntesis para realizar las operaciones involucradas.
Veamos algunos ejemplos.

Example (6.14)

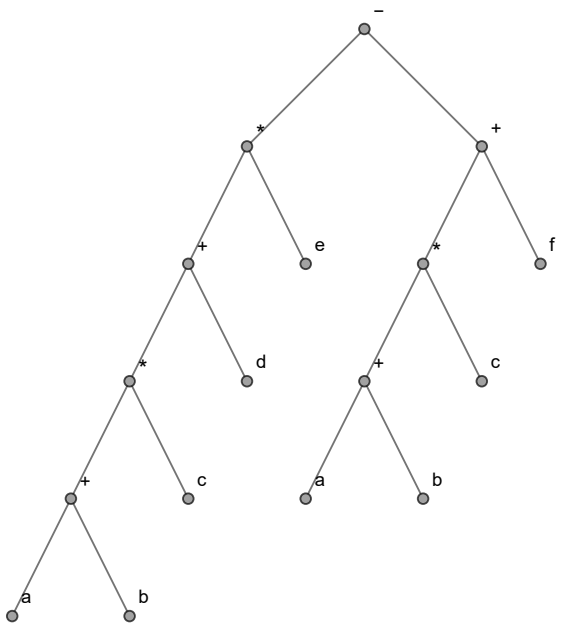
Determine una notación polaca y polaca inversa para la expresión algebraica del ejemplo 10.

Solución del ejemplo 6.14

En el ejemplo 10, la expresión algebraica del enunciado es:

$$(((a+b)\cdot c+d)\cdot e)-((a+b)\cdot c+f) \tag{29}$$

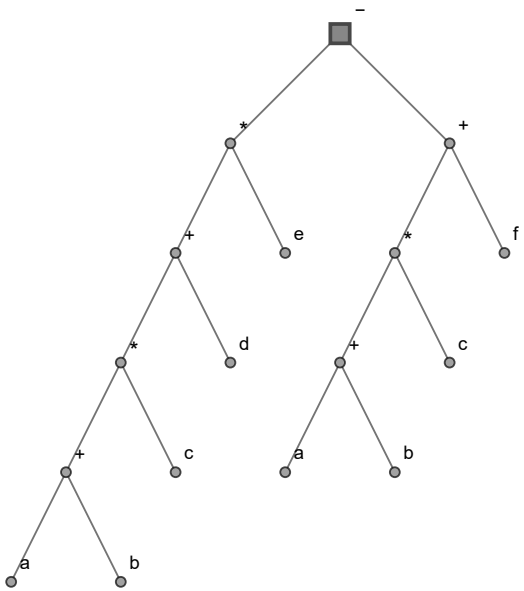
Y para ella, se construyó el siguiente árbol binario completo que la representa:

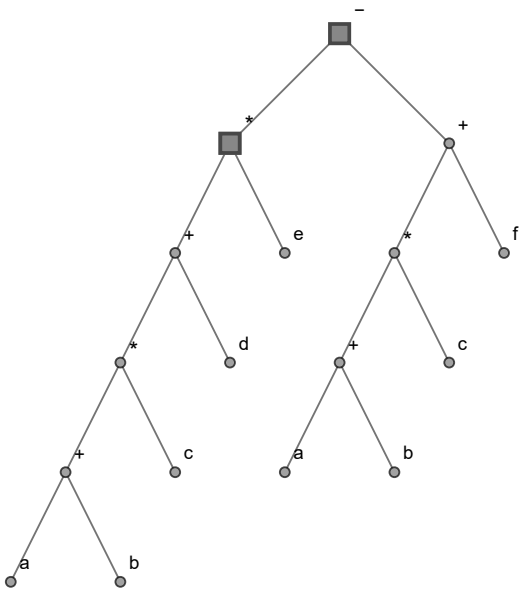


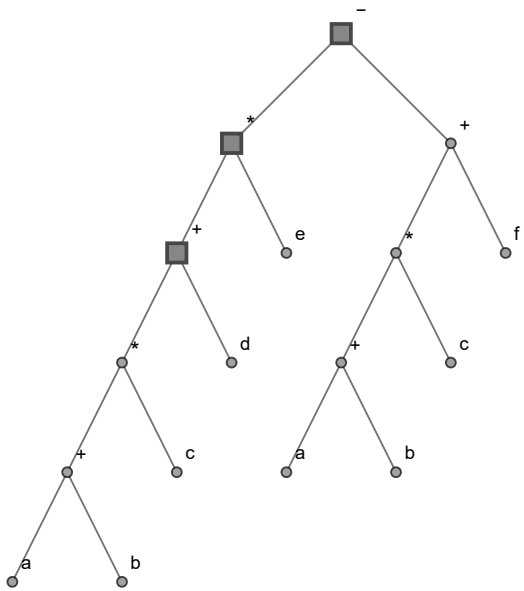
(30)

Solución del ejemplo 6.14

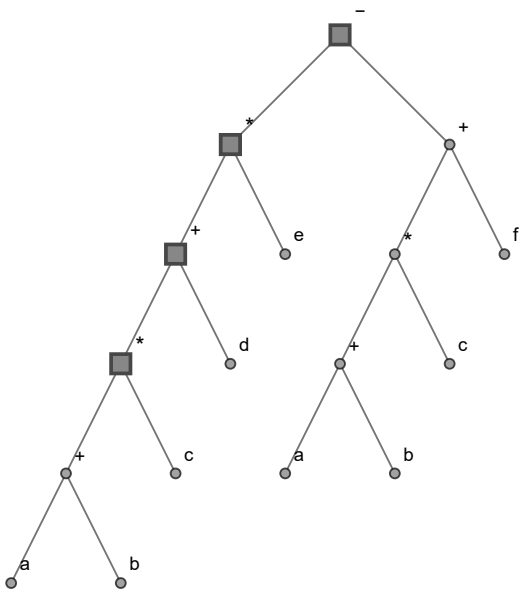
De acuerdo con la definición 20, una notación polaca de 29 se determina recorriendo el árbol 30 en orden inicial. La siguiente secuencia de imágenes muestra este recorrido (léase por filas):

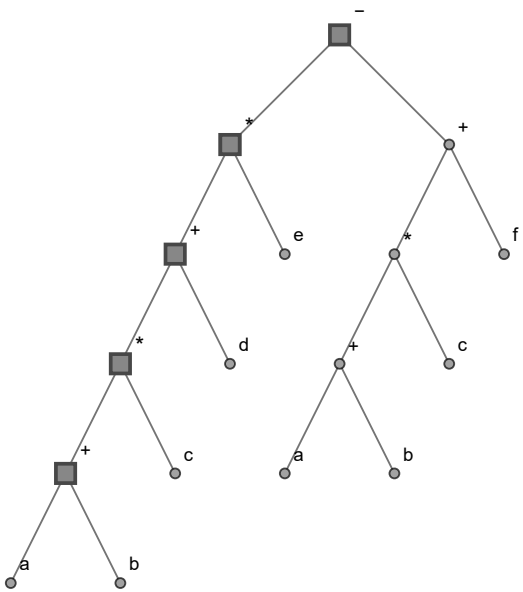


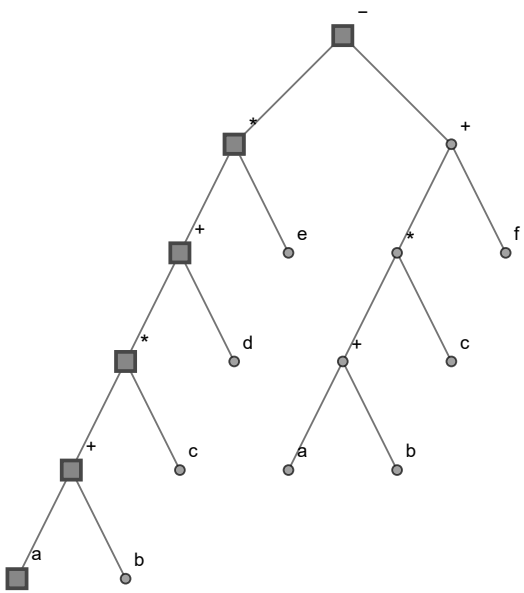




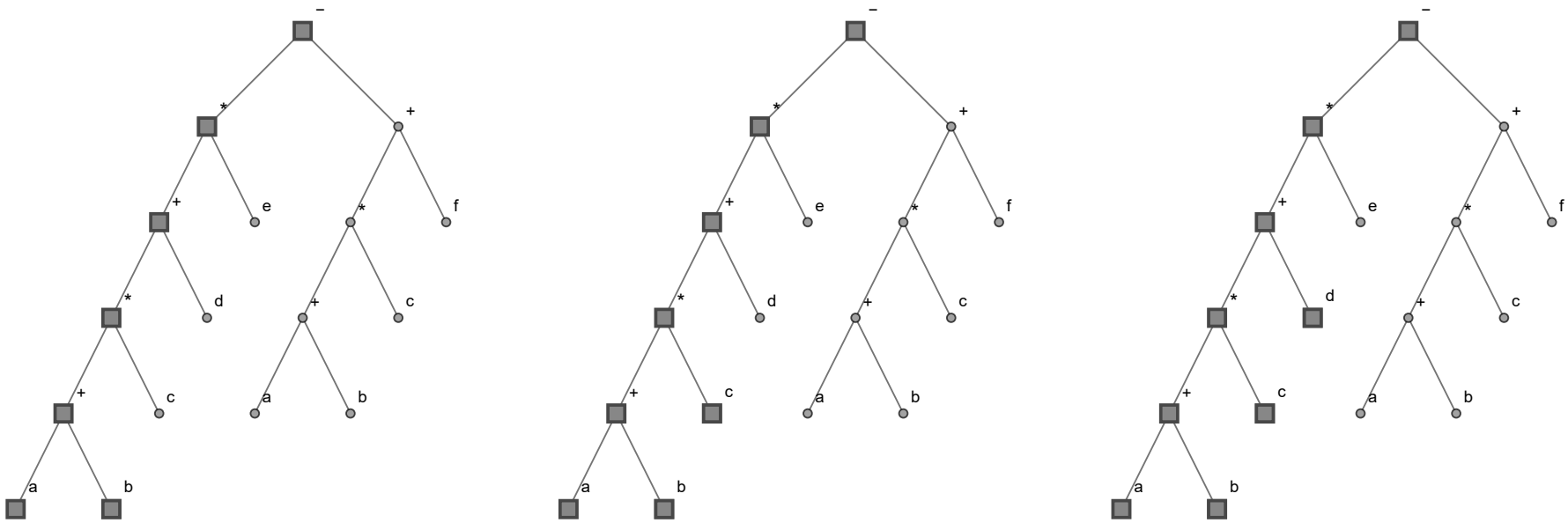
Solución del ejemplo 6.14



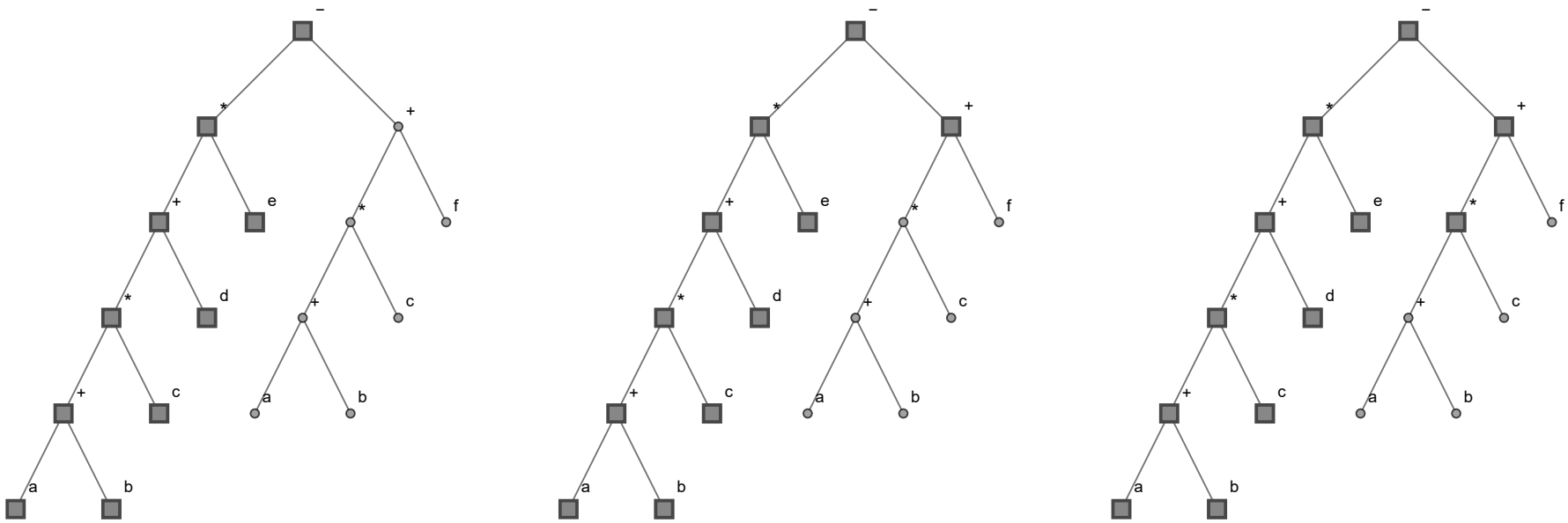




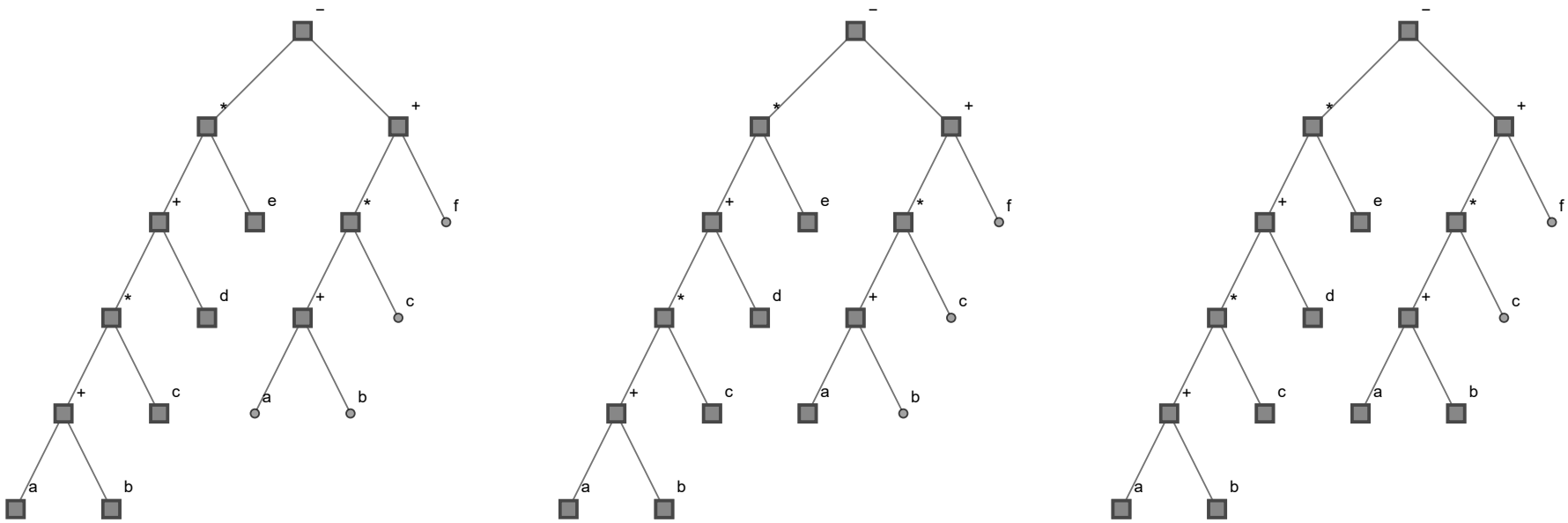
Solución del ejemplo 6.14



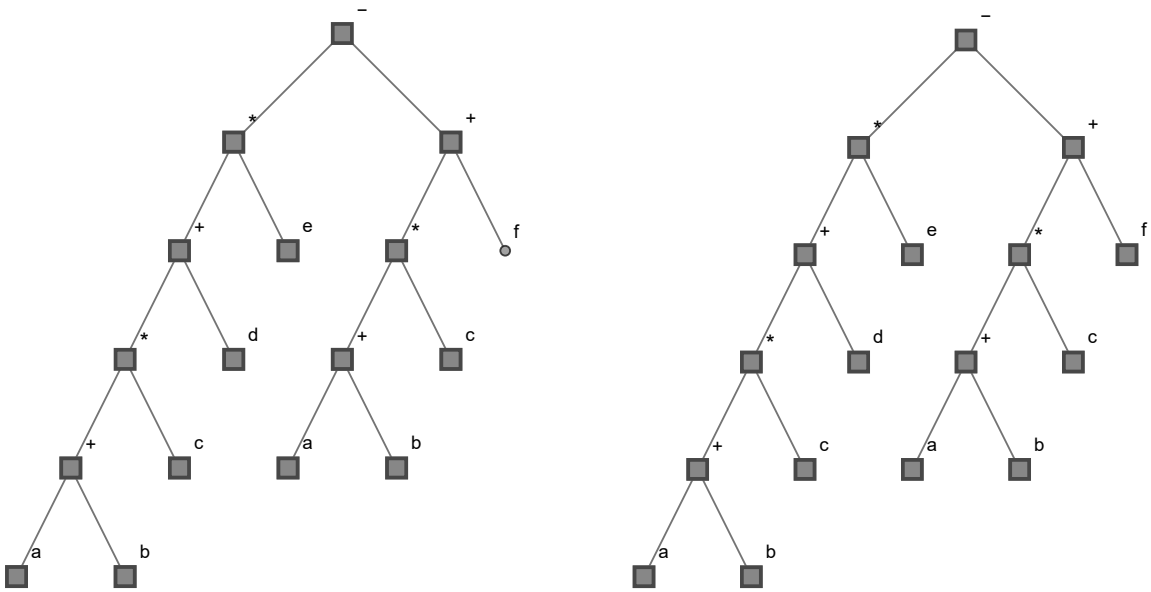
Solución del ejemplo 6.14



Solución del ejemplo 6.14



Solución del ejemplo 6.14

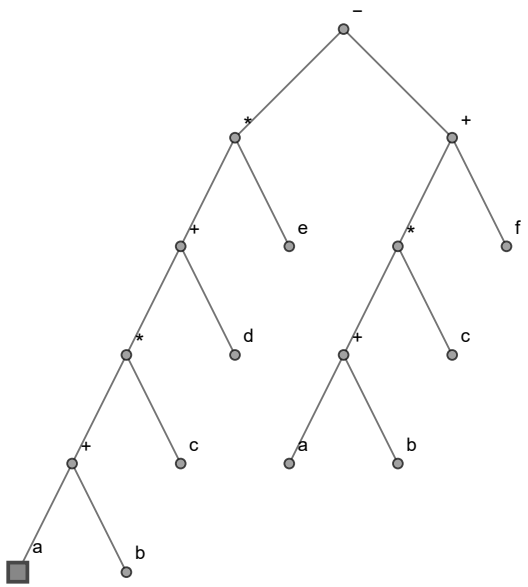


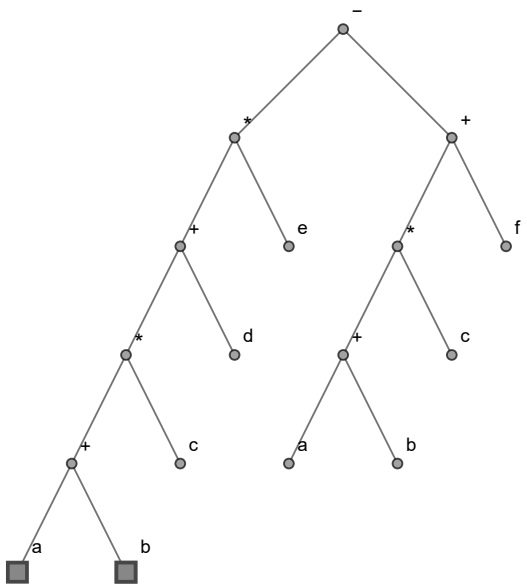
Se concluye que una notación polaca de 29 corresponde a:

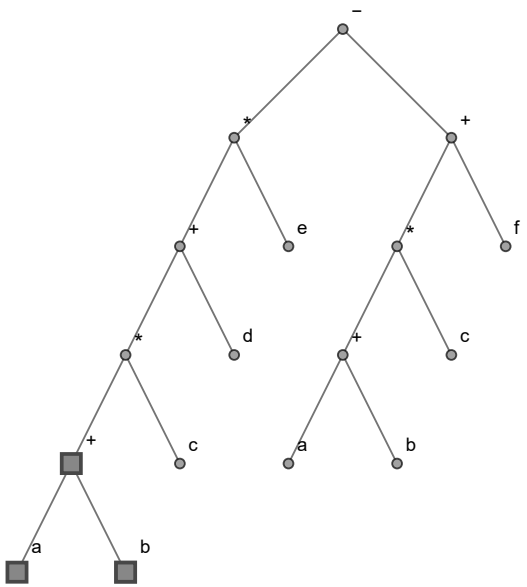
$$- * + * + abcde + * + abcf$$

Solución del ejemplo 6.14

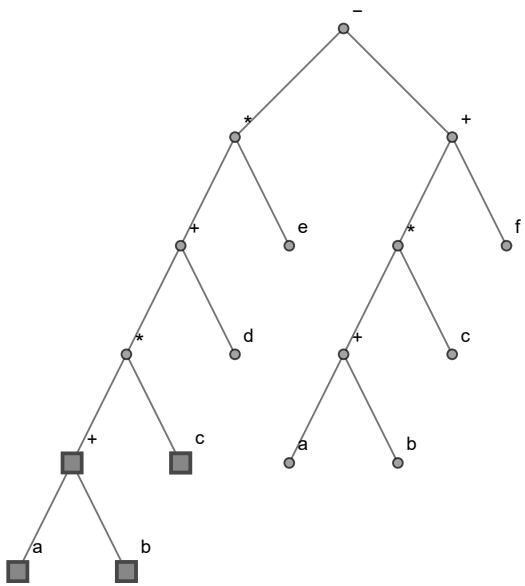
Por otra parte, si se recorre el árbol 30 en orden final, se obtiene una notación polaca inversa para la expresión algebraica 29, según la definición 20. La secuencia de imágenes compartida como sigue, describe el recorrido postfijo (léase por filas):

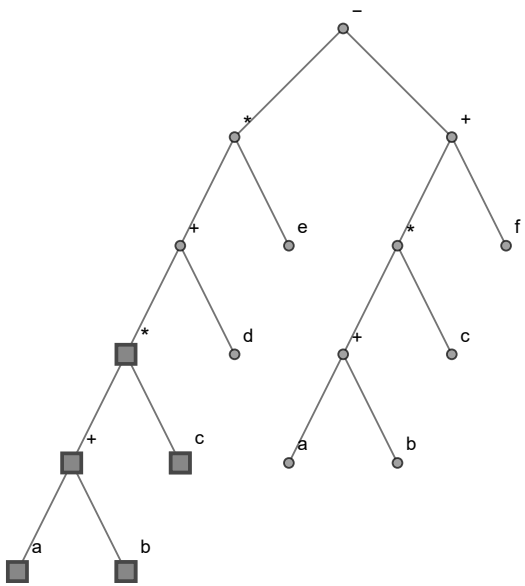


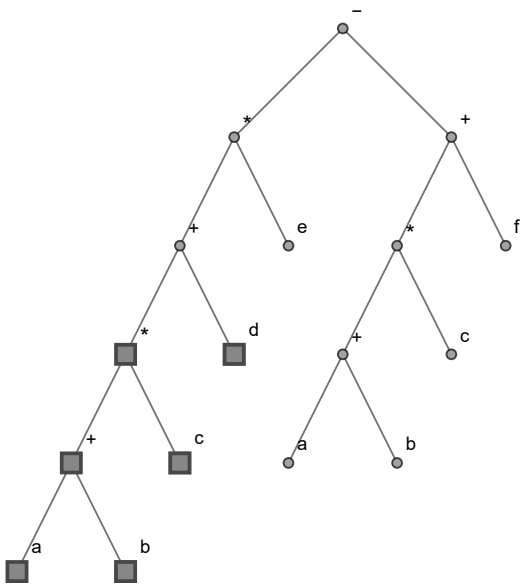




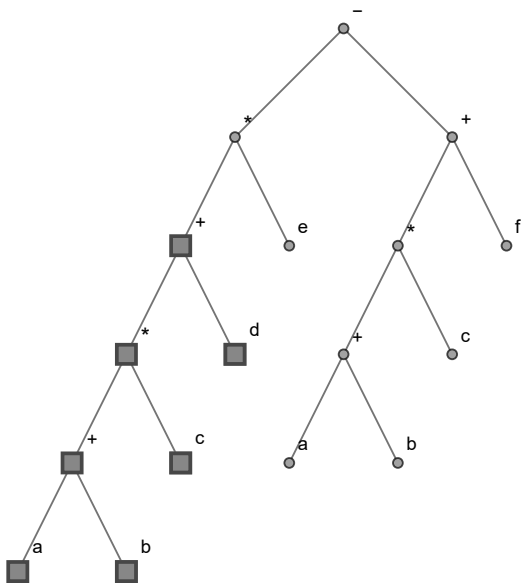
Solución del ejemplo 6.14

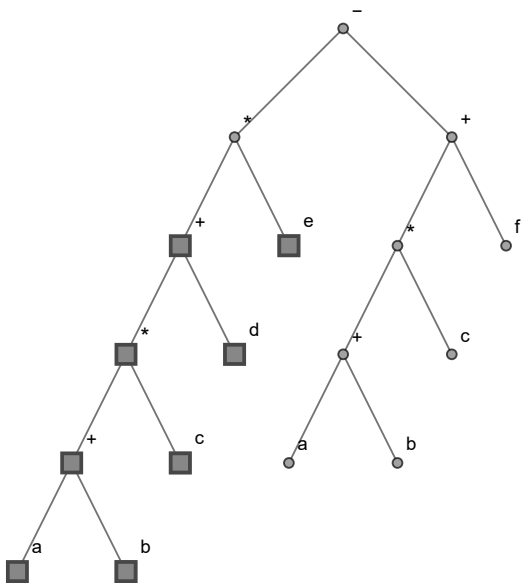


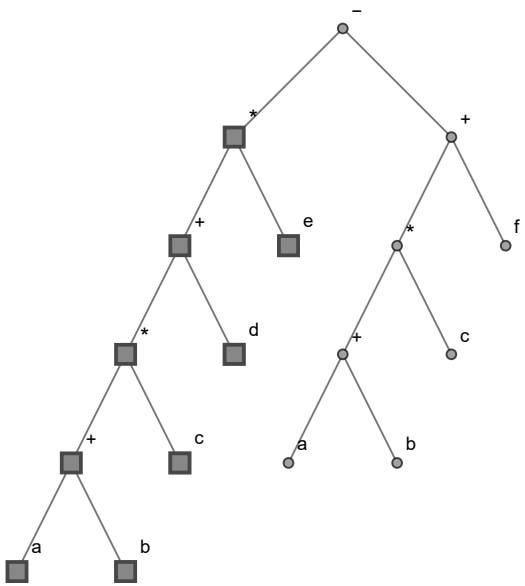




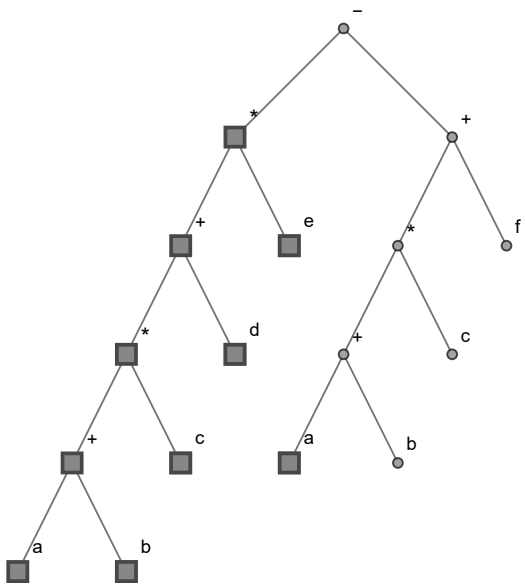
Solución del ejemplo 6.14

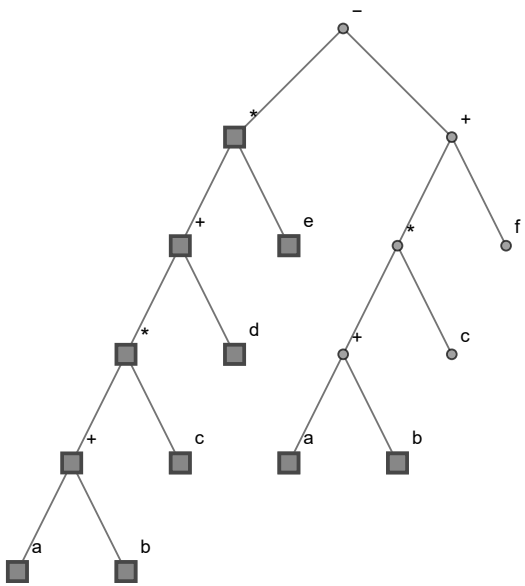


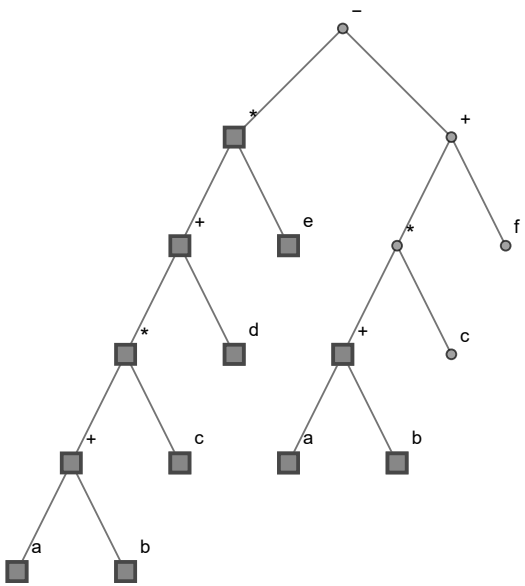




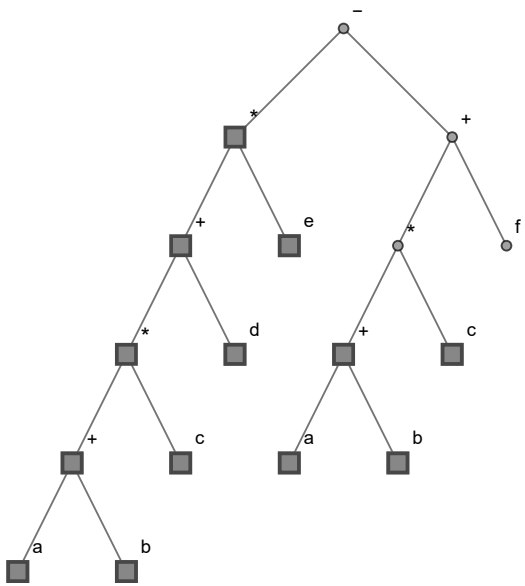
Solución del ejemplo 6.14

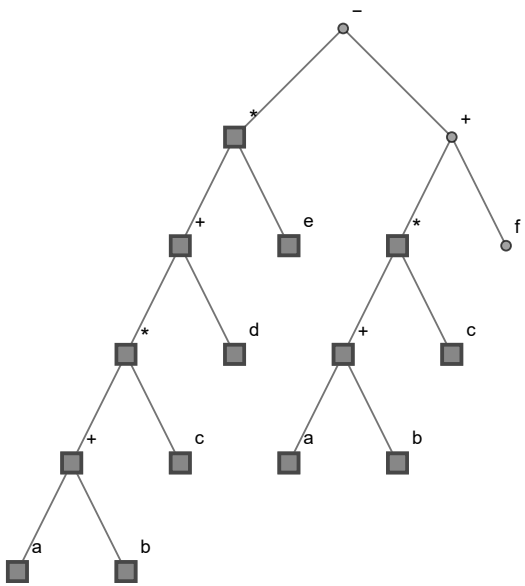


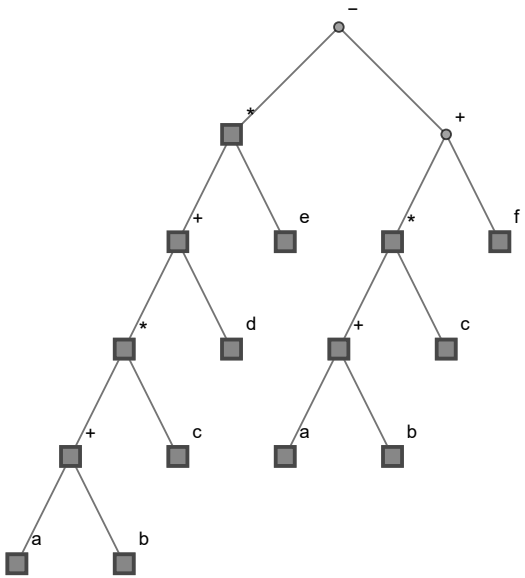




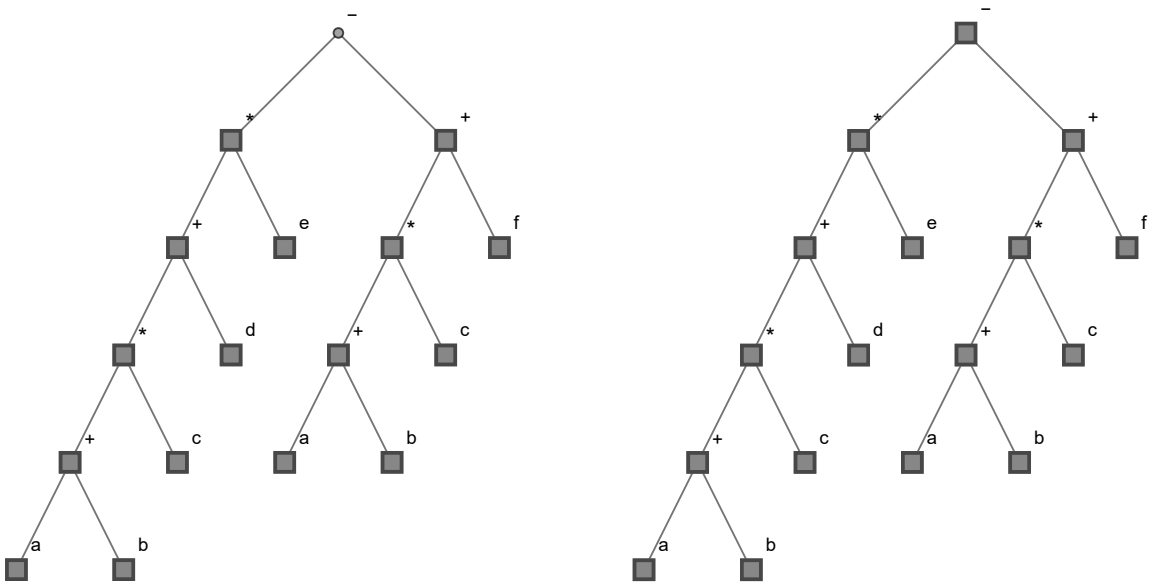
Solución del ejemplo 6.14







Solución del ejemplo 6.14



Luego, una notación polaca inversa de 29 es:

$$ab + c * d + e * ab + c * f + -$$

Nota

Si existen varios árboles binarios completos que representan una misma expresión algebraica, las formas notacionales polaca y polaca inversa no serán únicas. Por cada árbol distinto, se desprenderá una notación polaca y polaca inversa diferente.

Solución del ejemplo 6.14

La librería **VilCretas** integra las sentencias `Polaca` y `PolacaInversa` que retornan una notación polaca y polaca inversa, respectivamente, según un $k-$árbol de representación que *Mathematica* construye de manera automática. El árbol devuelto por el software no necesariamente será de orden $k = 2$. En este ejercicio:

In[ ] :=

```
Polaca[(((a + b) c + d) e) - ((a + b) c + f)]
PolacaInversa[(((a + b) c + d) e) - ((a + b) c + f)]
```

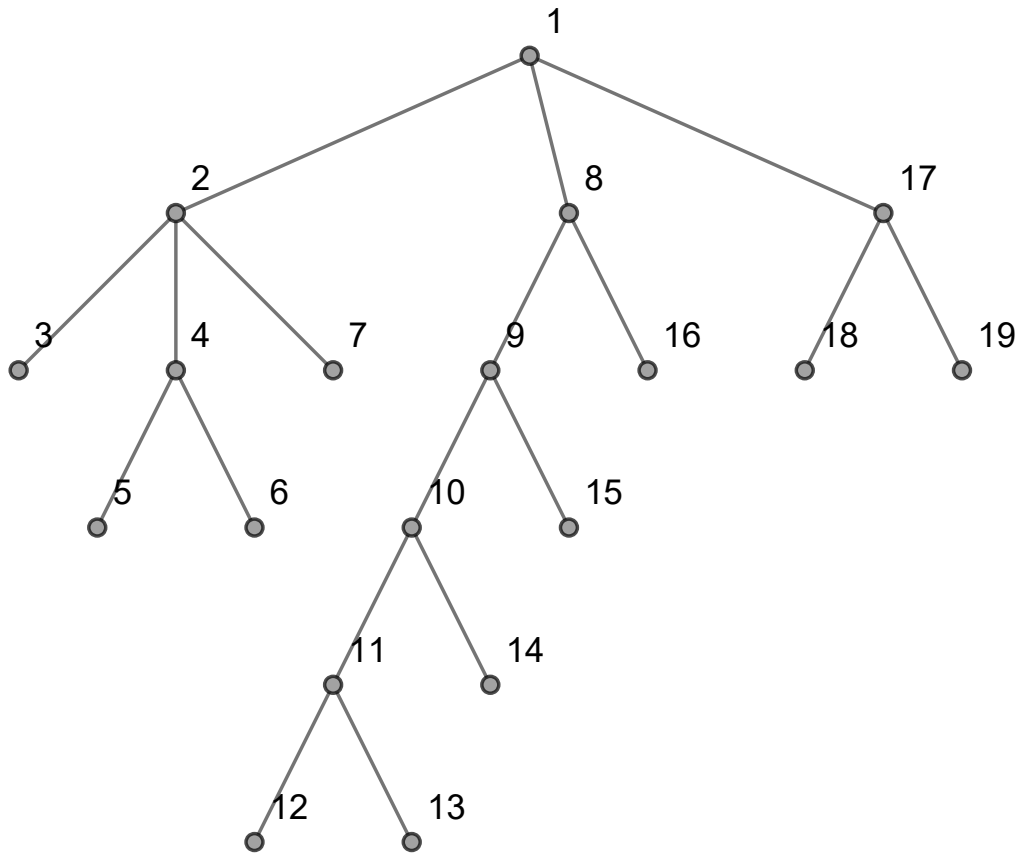
Solución del ejemplo 6.14

Se obtiene la siguiente salida:

Out[ ] =

Plus[Times[-1, Plus[a, b], c], Times[Plus[Times[Plus[a, b], c], d], e], Times[-1, f]]
{1->Plus, 2->Times, 3->-1, 4->Plus, 5->a, 6->b, 7->c, 8->Times, 9->Plus,
10->Times, 11->Plus, 12->a, 13->b, 14->c, 15->d, 16->e, 17->Times, 18->-1,
19->f}

Solución del ejemplo 6.14

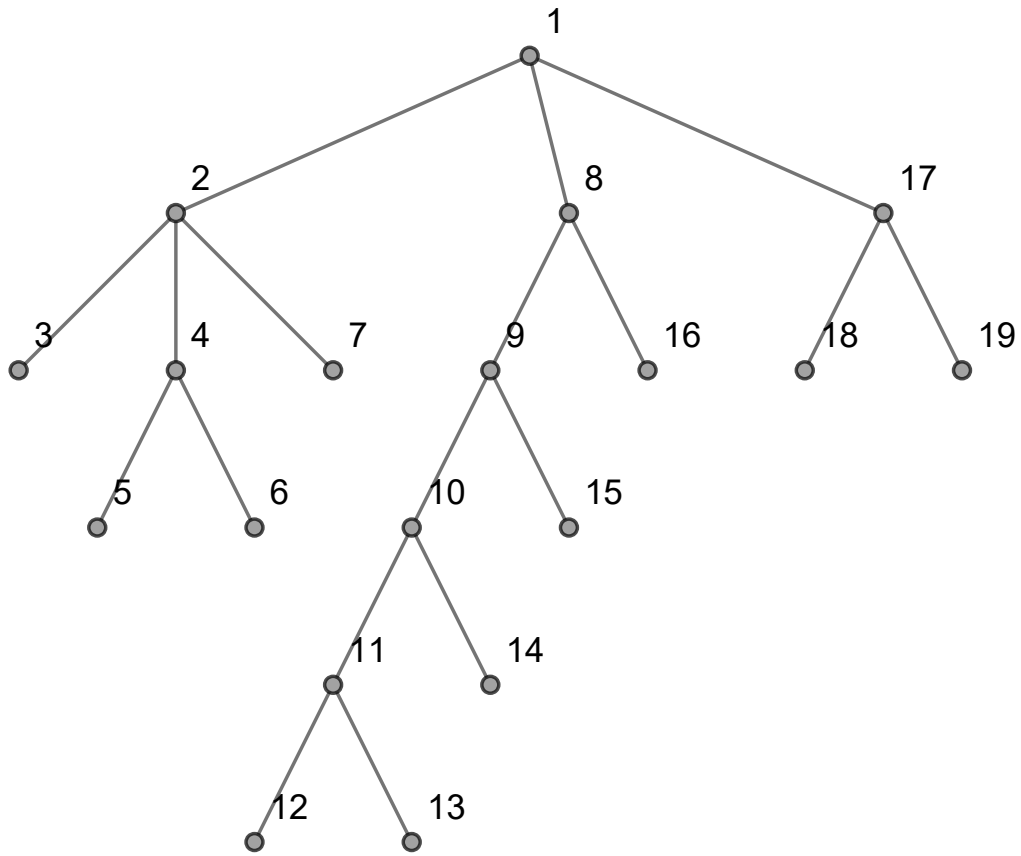


+*-1+abc*+*+abcde*-1f

Solución del ejemplo 6.14

Plus[Times[-1, Plus[a, b], c], Times[Plus[Times[Plus[a, b], c], d], e], Times[-1, f]]
{1->Plus, 2->Times, 3->-1, 4->Plus, 5->a, 6->b, 7->c, 8->Times, 9->Plus,
10->Times, 11->Plus, 12->a, 13->b, 14->c, 15->d, 16->e, 17->Times, 18->-1,
19->f}

Solución del ejemplo 6.14



-1ab+c*ab+c*d+e*-1f*+

Solución del ejemplo 6.14

En la salida algunos elementos se encuentran duplicados, incluyendo el árbol de representación de la expresión algebraica, dado que las sentencias `Polaca` y `PolacaInversa` comparten ciertos objetos de retorno. Claramente, las notaciones polaca y polaca inversa devueltas por *Mathematica* son diferentes a las encontradas sin ayuda del software, pues se está partiendo de árboles de representación distintos.

Descargue un archivo

`https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-150.zip`

Uso del comando Polaca

Explicación en video

https://youtu.be/GKvM-e1VIuo

Empleo de la instrucción `PolacaInversa`

Descargue un archivo

`https://youtu.be/4GlJ3S4Q6Ic`

Example (6.15)

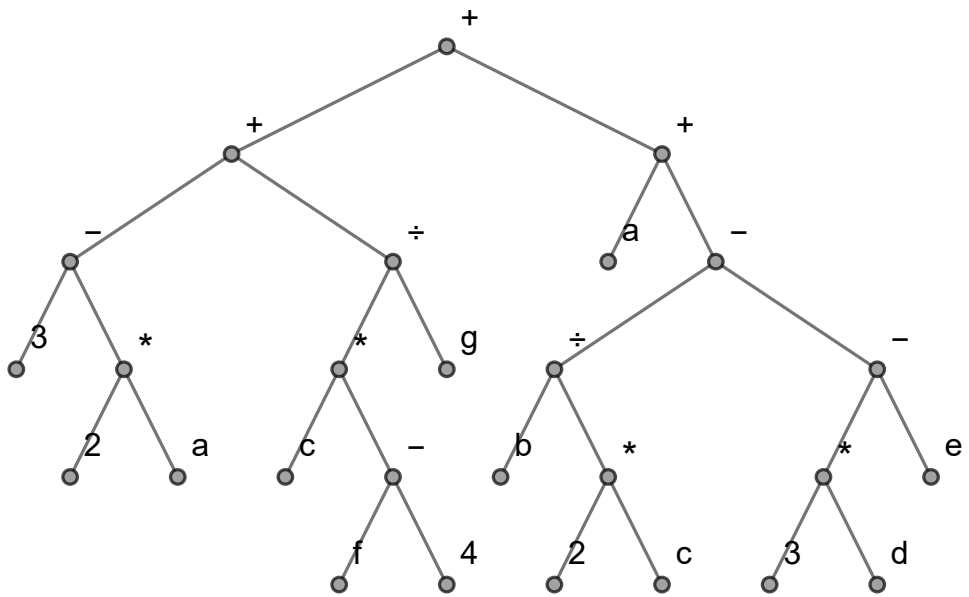
Halle una notación polaca y polaca inversa para la expresión algebraica del ejemplo 11.

Solución del ejemplo 6.15

En el ejemplo 11, la expresión algebraica de interés es:

$$(3-2a)+(b\div 2c-3d+e)+\frac{(f-4)\,c}{g}+a \tag{31}$$

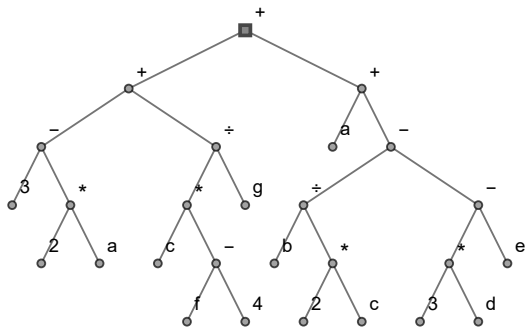
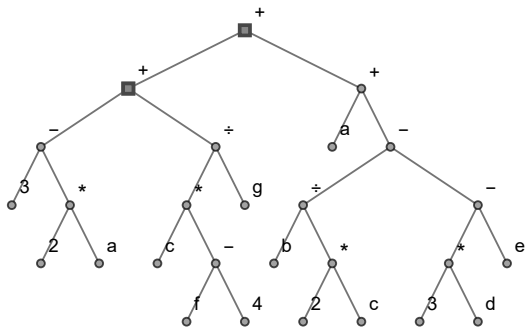
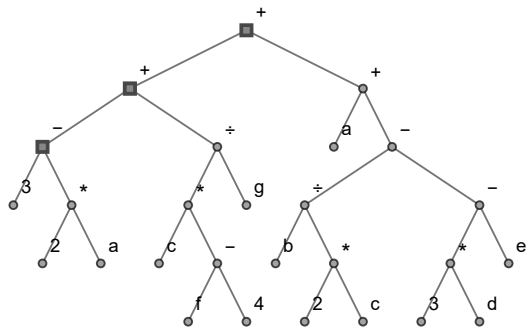
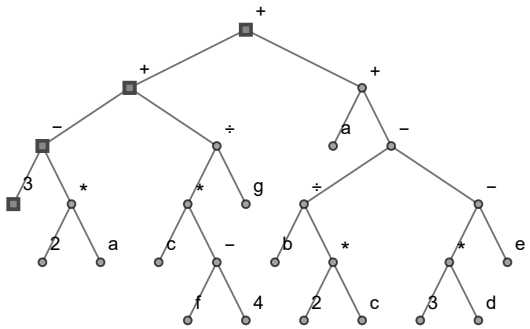
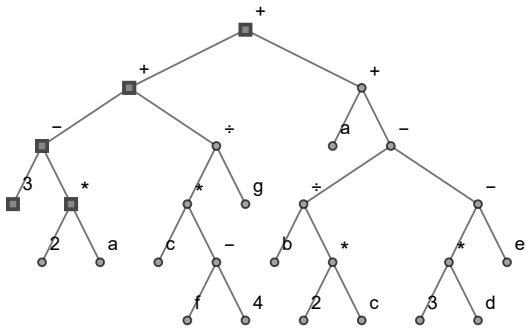
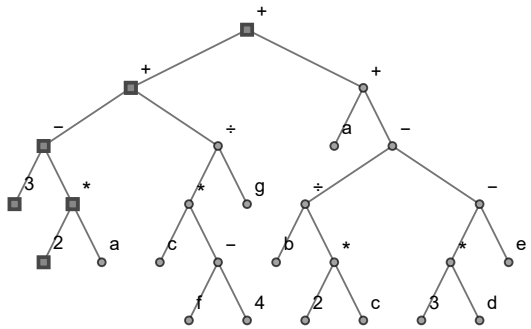
Donde se generó el siguiente árbol binario completo que la representa:



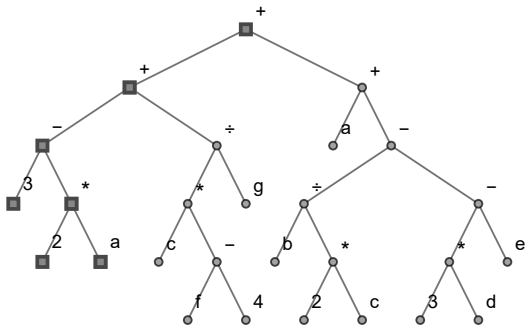
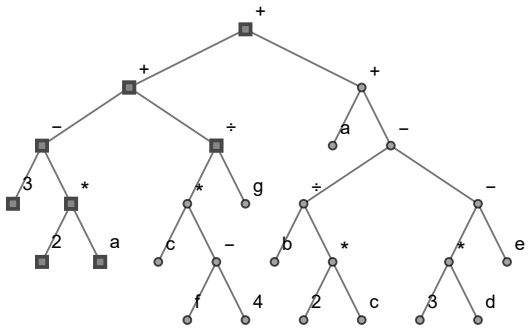
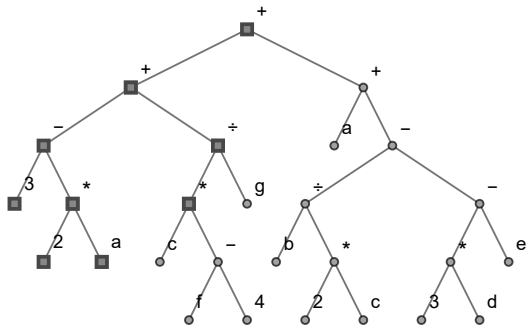
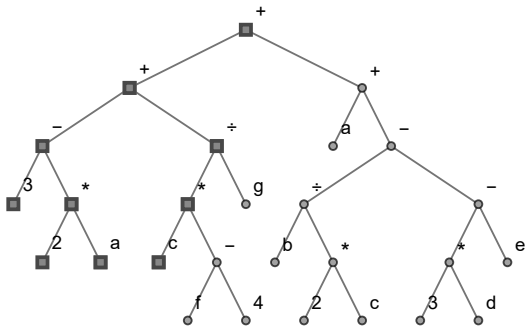
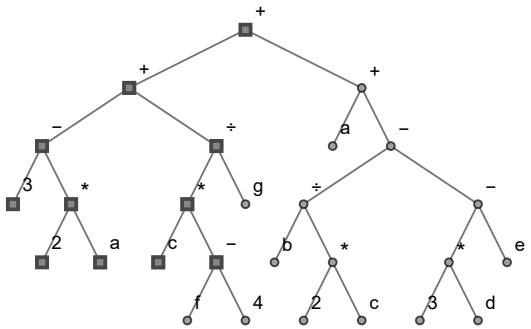
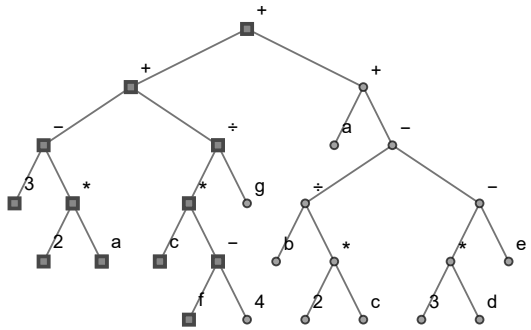
(32)

Solución del ejemplo 6.15

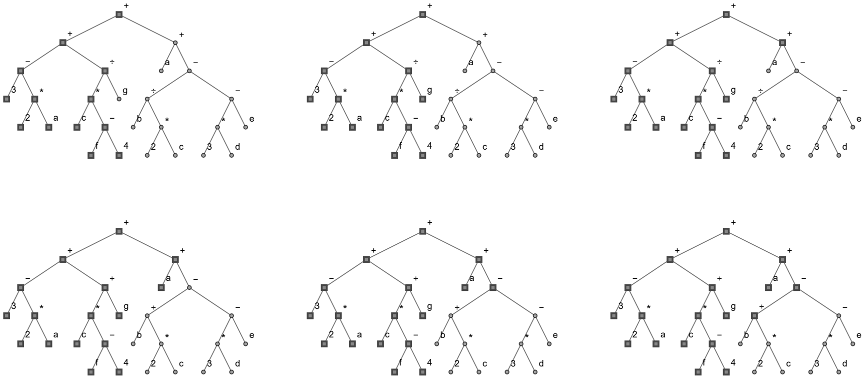
La notación polaca de la expresión algebraica 31 vinculada con el árbol 32, se halla al ejecutar un recorrido prefijo:

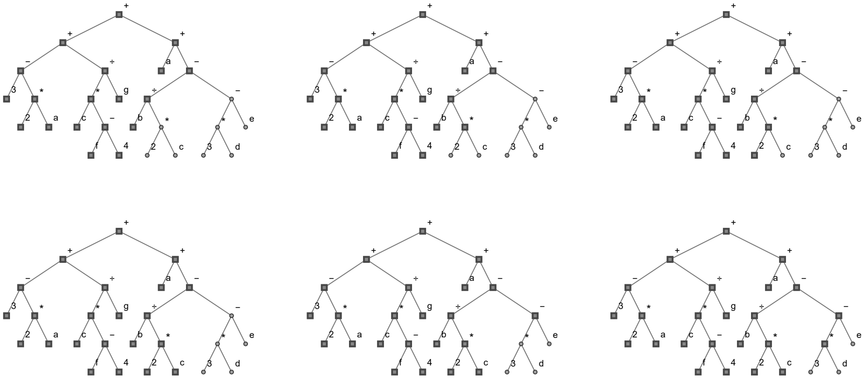
Solución del ejemplo 6.15

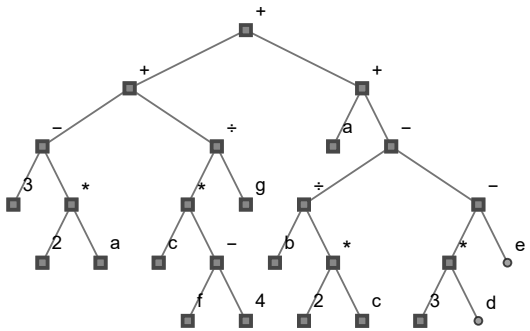
Solución del ejemplo 6.15

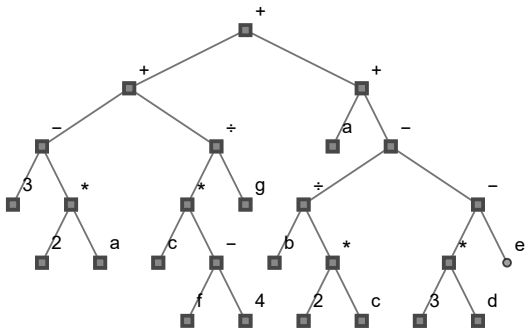


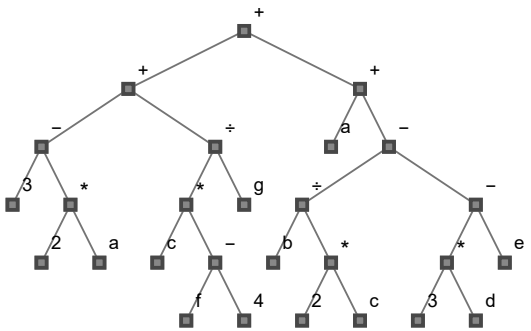
Solución del ejemplo 6.15



Solución del ejemplo 6.15





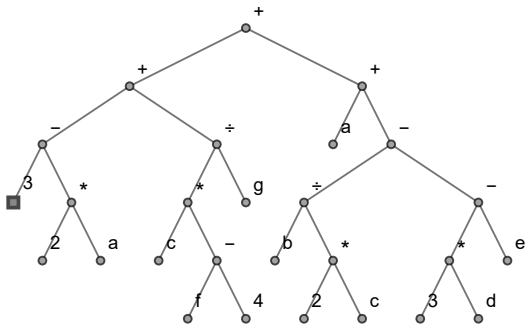
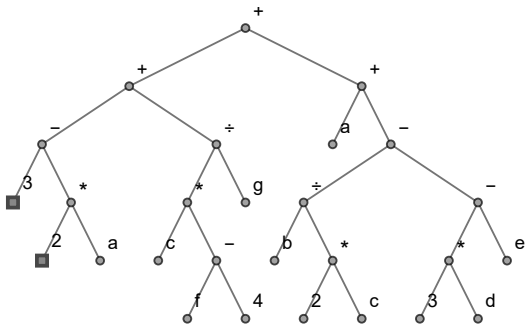
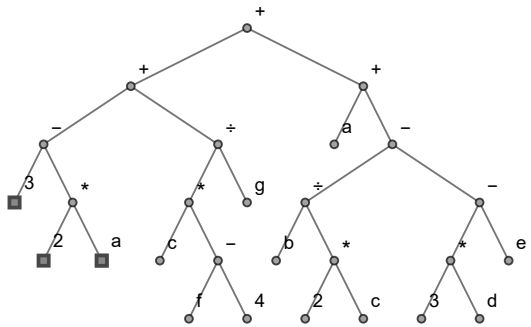
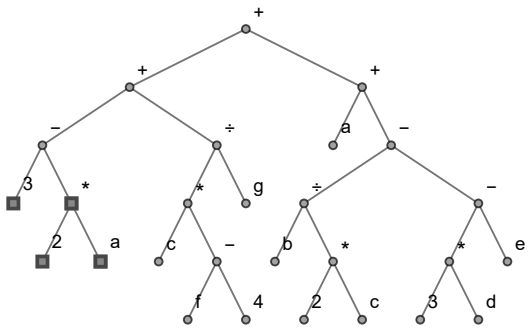
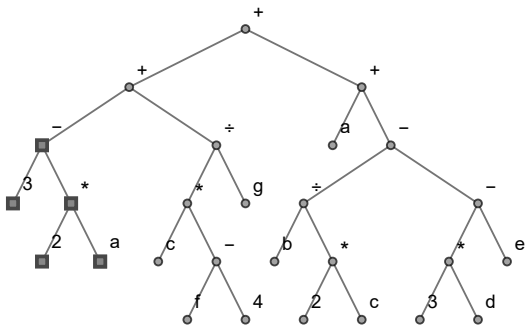
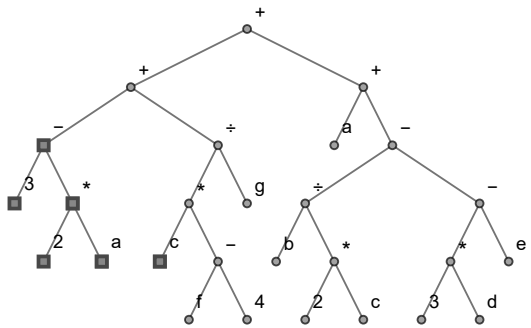


Como consecuencia, la notación polaca derivada es:

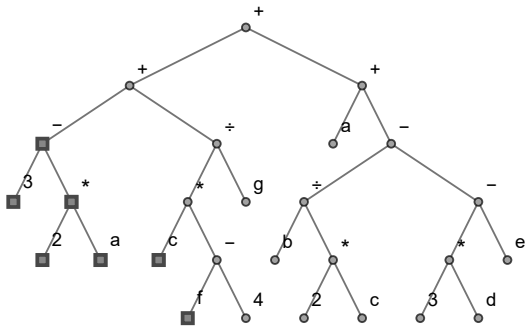
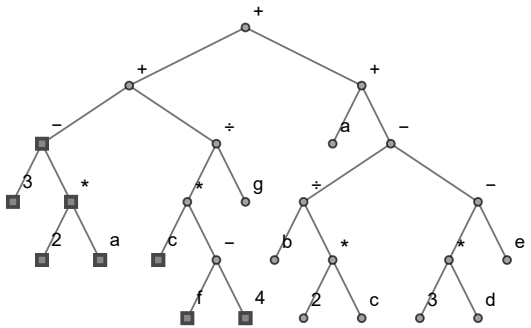
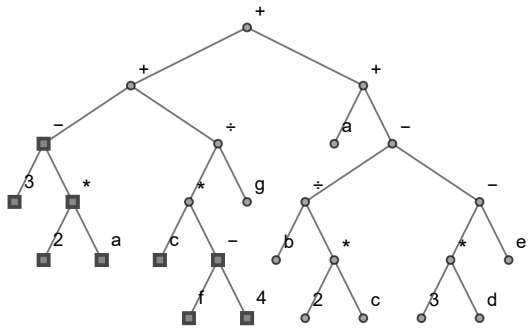
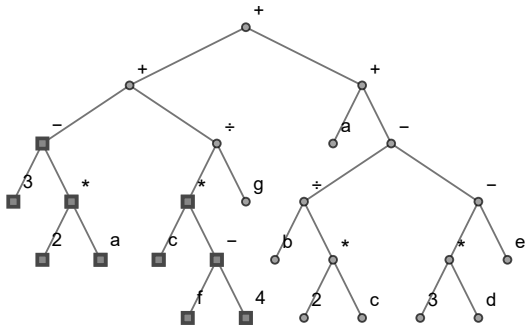
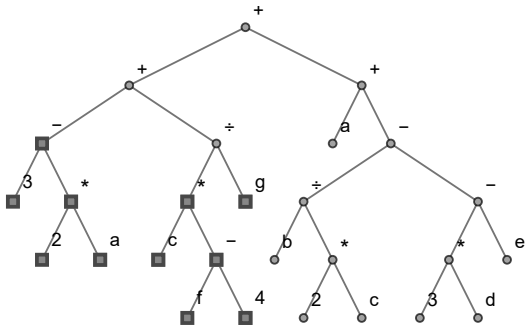
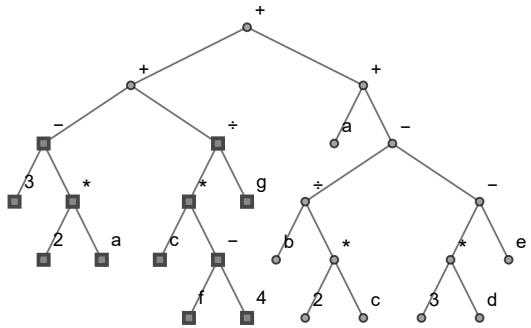
$$++-3*2a/*c-f4g+a-/b*2c-*3de$$

Solución del ejemplo 6.15

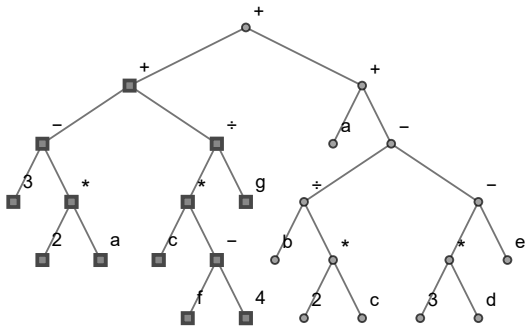
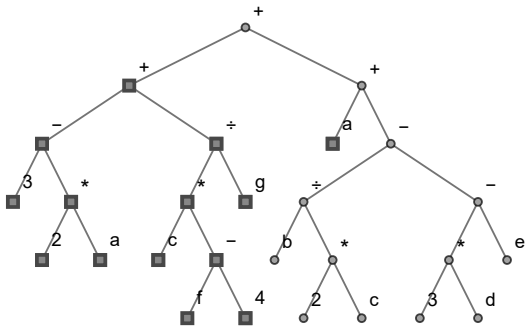
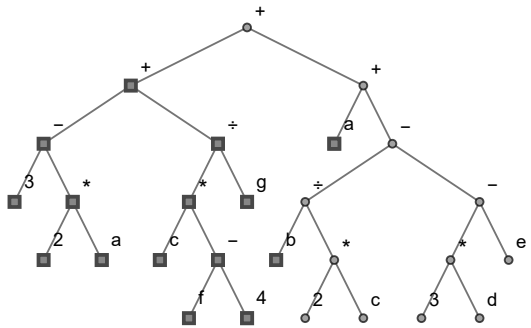
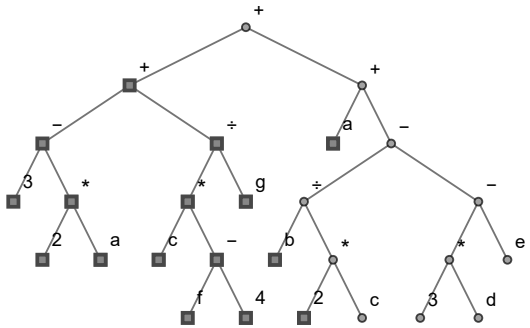
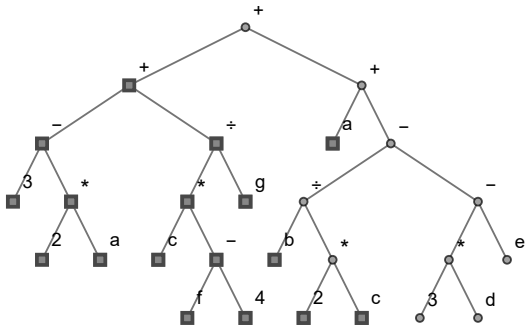
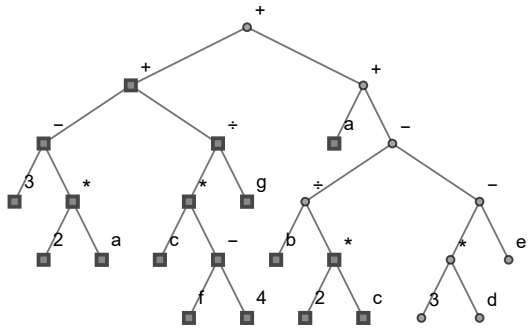
Ahora, la notación polaca inversa de la expresión 31 asociada al árbol 32, se resuelve al desarrollar un recorrido postfijo:

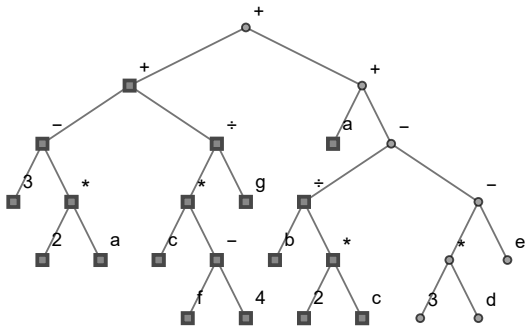
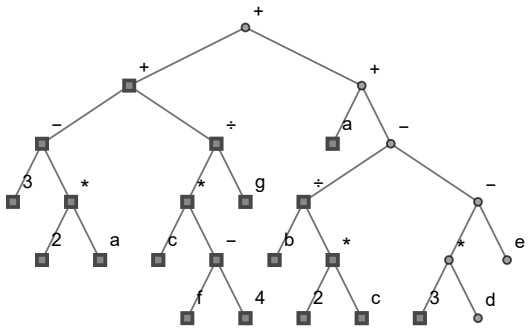
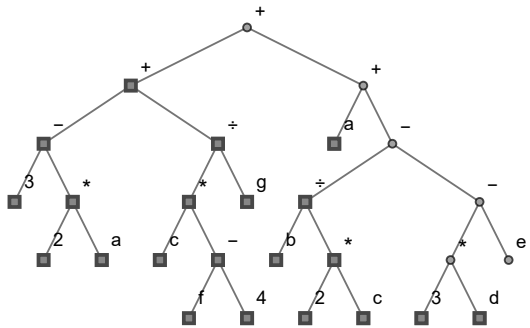
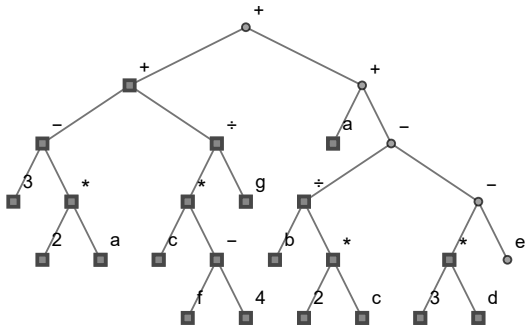
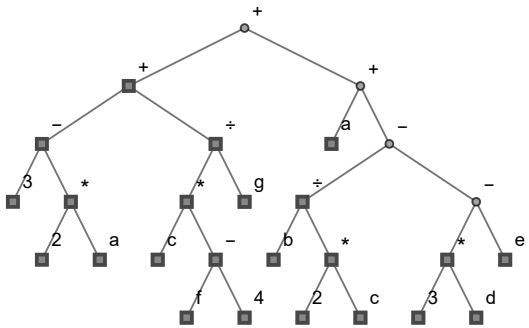
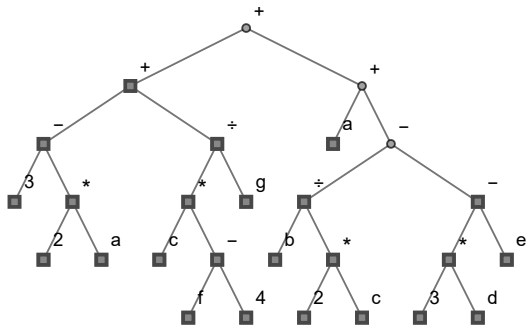
Solución del ejemplo 6.15

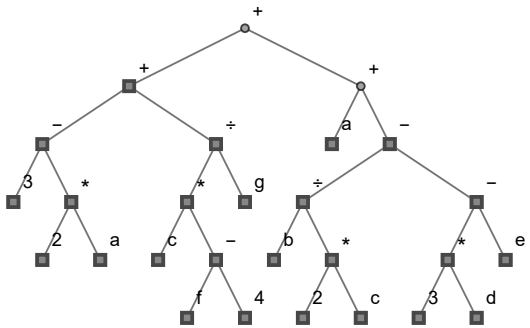
Solución del ejemplo 6.15

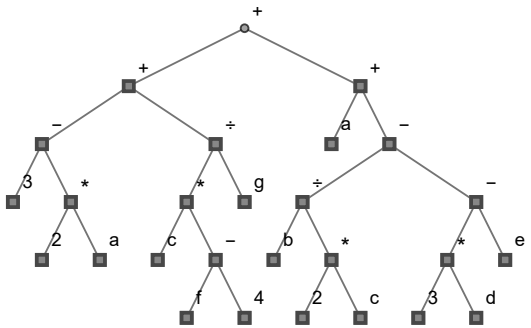







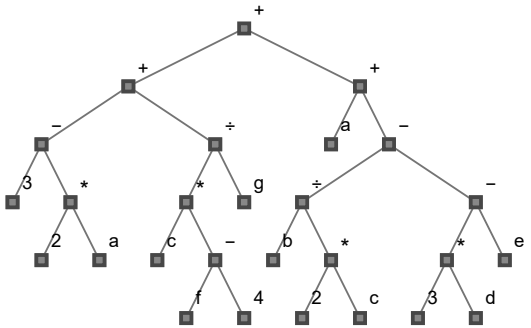
Solución del ejemplo 6.15

Solución del ejemplo 6.15







Por ende, la notación polaca inversa obtenida es:

$$32a*-cf4-*g/+ab2c*/3d*e--++$$

Solución del ejemplo 6.15

Las dos notaciones encontradas se pueden verificar recurriendo a los comandos `Prefijo` y `Postfijo`:

In[ ] :=

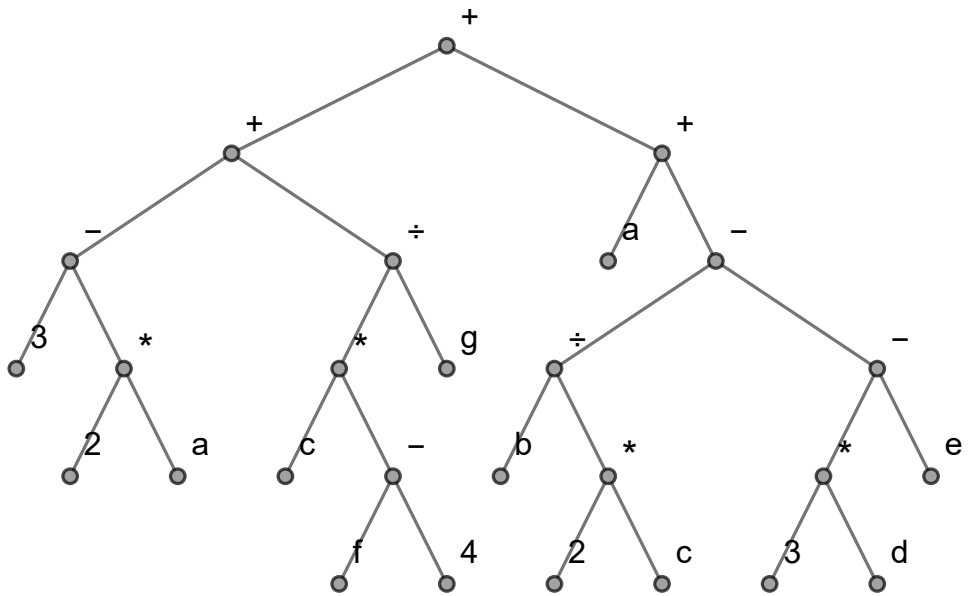
```
arbol = Graph[{{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6},
{3, 7}, {4, 8}, {4, 9}, {5, 10}, {5, 11}, {7, 12}, {7, 13},
{9, 14}, {9, 15}, {10, 16}, {10, 17}, {12, 18}, {12, 19},
{13, 20}, {13, 21}, {17, 22}, {17, 23}, {19, 24}, {19, 25},
{20, 26}, {20, 27}}, VertexLabels -> {1 -> ''+'', 2 ->
''+'', 3 -> ''+'', 4 -> ''-'', 5 -> ''÷'', 6 -> ''a'', 7
-> ''-'', 8 -> ''3'', 9 -> ''*'', 10 -> ''*'', 11 ->
''g'', 12 -> ''÷'', 13 -> ''-'', 14 -> ''2'', 15 ->
''a'', 16 -> ''c'', 17 -> ''-'', 18 -> ''b'', 19 ->
''*'', 20 -> ''*'', 21 -> ''e'', 22 -> ''f'', 23 ->
''4'', 24 -> ''2'', 25 -> ''c'', 26 -> ''3'', 27 ->
''d''}]
```

Solución del ejemplo 6.15

```
StringJoin[Prefijo[arbol, 1] /. {1 -> ''+'', 2 -> ''+'', 3
-> ''+'', 4 -> ''-'', 5 -> ''÷'', 6 -> ''a'', 7 ->
''-'', 8 -> ''3'', 9 -> ''*'', 10 -> ''*'', 11 -> ''g'',
12 -> ''÷'', 13 -> ''-'', 14 -> ''2'', 15 -> ''a'', 16
-> ''c'', 17 -> ''-'', 18 -> ''b'', 19 -> ''*'', 20 ->
''*'', 21 -> ''e'', 22 -> ''f'', 23 -> ''4'', 24 ->
''2'', 25 -> ''c'', 26 -> ''3'', 27 -> ''d''}]
StringJoin[Postfijo[arbol, 1] /. {1 -> ''+'', 2 -> ''+'',
3 -> ''+'', 4 -> ''-'', 5 -> ''÷'', 6 -> ''a'', 7 ->
''-'', 8 -> ''3'', 9 -> ''*'', 10 -> ''*'', 11 -> ''g'',
12 -> ''÷'', 13 -> ''-'', 14 -> ''2'', 15 -> ''a'', 16
-> ''c'', 17 -> ''-'', 18 -> ''b'', 19 -> ''*'', 20 ->
''*'', 21 -> ''e'', 22 -> ''f'', 23 -> ''4'', 24 ->
''2'', 25 -> ''c'', 26 -> ''3'', 27 -> ''d''}]
```

Solución del ejemplo 6.15

Out[ ] =



++-3*2a/*c-f4g+a-/b*2c-*3de

32a*-cf4-*g/+ab2c*/3d*e--++

Solución del ejemplo 6.15

En el **In[ ]**, el árbol se ha construido con ayuda de un comando propio de *Wolfram Mathematica* llamado: `Graph`. No se utilizó la sentencia `Arbol` o `ArbolR`, pues `Graph` cuenta con la opción `VertexLabels` capaz de añadir etiquetas a los vértices de un grafo. Este atributo resulta indispensable para poder diseñar en *Mathematica* el árbol 32. También, el comando `StringJoin` observado en el **In[ ]** es otra instrucción propia de *Wolfram* que une varios "strings" en una sola hilera y en este caso, resulta de utilidad en la elaboración final de las notaciones polaca y polaca inversa. Por otra parte, el operador `/.` se emplea en el **In[ ]**, con el objetivo de proceder a la sustitución de los nodos del grafo por sus etiquetas correspondientes.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-151.zip

- A partir de un grafo $G = (V, E)$ conexo es posible construir un árbol que contiene cada uno de los vértices de G. A estos árboles se les llama "generadores". Los métodos para obtener un árbol generador merecen una atención especial. La próxima sección aborda este importante tema.

Árboles generadores

Prof. Enrique Vílchez Quesada

Universidad Nacional de Costa Rica

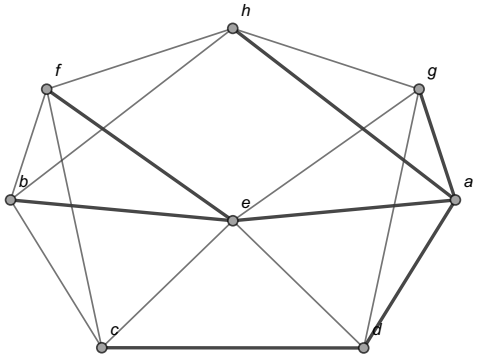
Definición 6.3

Iniciaremos con el concepto de "árbol generador" o de "expansión".

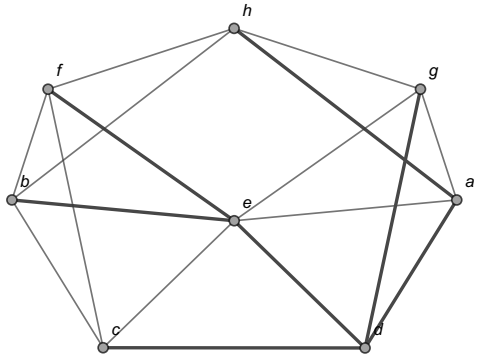
Definition (6.3)

Sea $G = (V, E)$ un grafo no dirigido conexo, se llama árbol de expansión, generador o recubridor de G, a un árbol subgrafo de G, que contiene todos sus vértices.

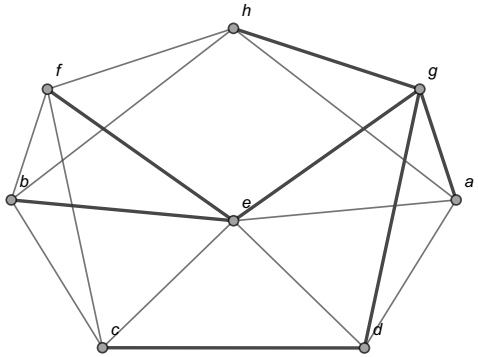
La figura 4 comparte cuatro ejemplos de distintos árboles de expansión sobre un mismo grafo conexo.



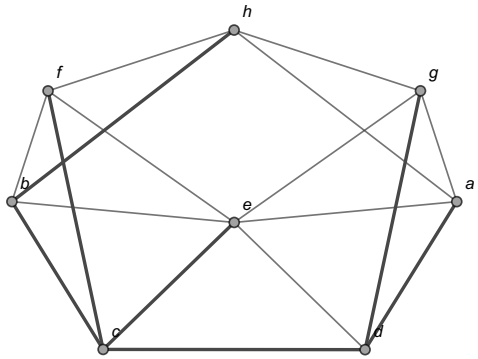
(a) Árbol de expansión con raíz a



(b) Árbol de expansión con raíz d



(c) Árbol de expansión con raíz g



(d) Árbol de expansión con raíz c

Figura: Ejemplos de árboles generadores

Teorema 6.5

Como el lector preverá de la definición 23, un grafo G no dirigido tiene un árbol de expansión cuando éste es conexo. El siguiente teorema enuncia esta propiedad.

Theorem (6.5)

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

En esta teoría, existen dos algoritmos para determinar árboles de expansión sobre un grafo G conexo, no necesariamente ponderado. Estos algoritmos se llaman: "buscar primero a lo ancho" y "buscar primero a lo largo". Veamos.

Teorema 6.6. Buscar primero a lo ancho

Theorem (6.6)

Sea $G = (V, E)$ un grafo conexo con $V = \{v_1, v_2, \ldots, v_n\}$ el conjunto de vértices de G en ese orden, entonces:

1. *Considere el conjunto $C = \{v_1\}$ y T el árbol generador a construir, inicializado con la raíz v_1.*
2. *Para cada $w \in C$, añada a T todas las aristas $[w, v]$ de G, con v un vértice de G en el orden de V y donde $[w, v]$ 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 C, por los hijos en T de cada uno de los nodos que se encuentran en C, siguiendo el orden de V. Vaya al paso 2.*

Teorema 6.7. Buscar primero a lo largo

Theorem (6.7)

Sea $G = (V, E)$ un grafo conexo con $V = \{v_1, v_2, \ldots, v_n\}$ el conjunto de vértices de G en ese orden, entonces:

1. *Considere $w = v_1$ y T el árbol generador a construir, inicializado con la raíz v_1.*
2. *Añada a T una arista $[w, v_j]$ de G, con v_j un vértice de G con j mínimo, 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, continúe con el paso 3.*
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 regrese al paso 2.*

Comentario sobre los teoremas 25 y 26

En los teoremas 25 y 26 se utiliza el término "buscar" en su denominación, pues cuando el grafo G es un árbol T, estos métodos permiten efectuar búsquedas sobre T. En la sección anterior, se señaló que los recorridos prefijo, interfijo y postfijo constituyen también, algoritmos de búsqueda. La diferencia entre estos dos grupos de procedimientos, radica en que los algoritmos a lo "ancho" y a lo "largo" admiten cualquier orden k, $k \in \mathbb{N}$, en el árbol T correspondiente a la estructura de datos.
Los algoritmos "buscar primero a lo ancho" y "buscar primero a lo largo", por consiguiente, tienen un doble propósito: si el grafo G recibido no es un árbol, los métodos a lo "ancho" y a lo "largo" se encargan de devolver un árbol de expansión sobre G y si G satisface lo contrario, los procedimientos a lo "ancho" y a lo "largo" proveen un orden con la finalidad de buscar un dato en los nodos del grafo.

Comentario sobre los teoremas 25 y 26

Cabe señalar que si G es un árbol T, el retorno de "buscar primero a lo ancho" y "buscar primero a lo largo" será el mismo grafo G inicial, por lo que, en dicho caso estos algoritmos cobran sentido solo para realizar búsquedas.

También es indispensable mencionar que en los métodos a lo "ancho" y a lo "largo", si T posee $n-1$ aristas de G se ha finalizado el proceso, con n el número de vértices del grafo. La justificación de ello obedece al teorema 3, donde se estableció que todo árbol siempre contiene una cantidad de lados igual a $n-1$.

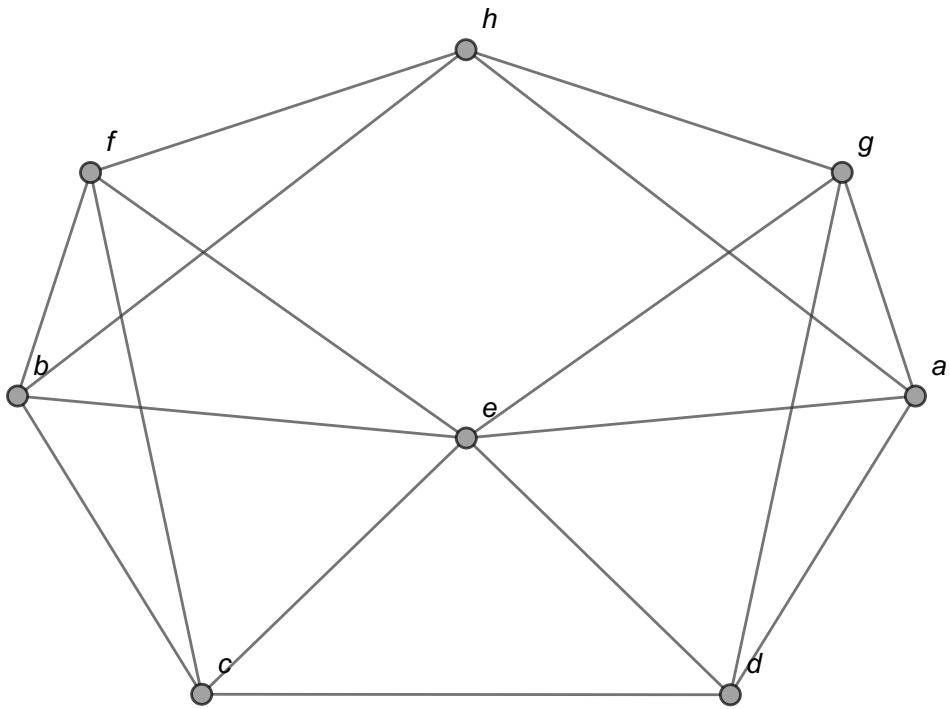
Se aclara al alumno, además, que en algunos contextos al algoritmo "buscar primero a lo largo", se le llama también, "buscar primero a profundidad".

Finalmente, en este libro se utilizan los acrónimos **BPA** y **BPL** para hacer referencia a los procedimientos a lo "ancho" y a lo "largo", respectivamente.

- Se abordarán algunos ejemplos al respecto, sobre el uso de los algoritmos "buscar primero a lo ancho" y "buscar primero a lo largo". En los ejercicios propuestos los grafos empleados no corresponden a árboles, sin embargo, se insiste al estudiante que si así fuera, el objetivo se centraría en ejecutar la búsqueda de un dato y no en determinar un árbol recubridor.

Example (6.16)

Encuentre un árbol generador en el grafo conexo dado, empleando los algoritmos **BPA** y **BPL**, asumiendo el orden del abecedario para el conjunto V de vértices, es decir, $V = \{a, b, c, d, e, f, g, h\}$.



Solución del ejemplo 6.16

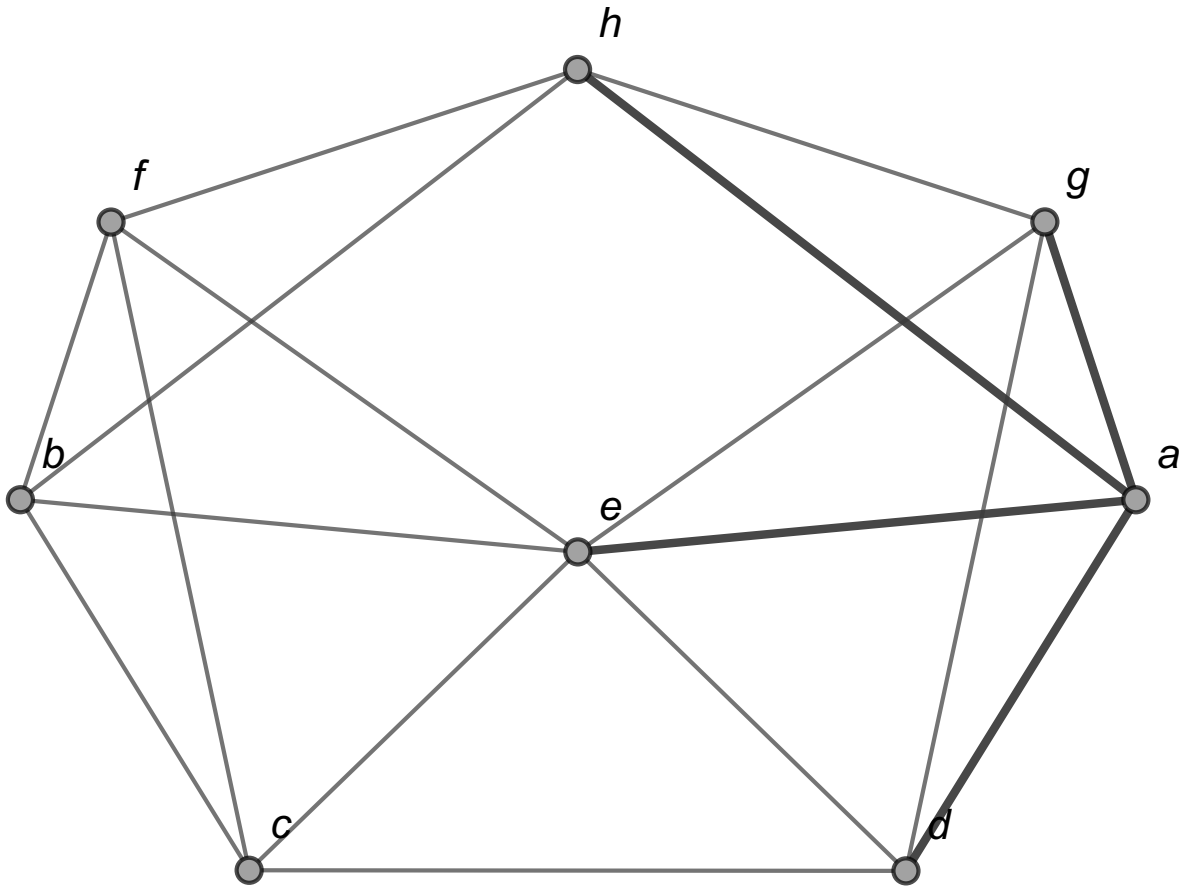
En el presente ejercicio el número de nodos n del grafo es $n = 8$, en consecuencia, los algoritmos **BPA** y **BPL** finalizan cuando el árbol T a construir, contenga $n - 1 = 8 - 1 = 7$ aristas.

Se comenzará la resolución del ejemplo, aplicando el método "buscar primero a lo ancho". Se parte del nodo a como primer elemento de V y se agregan al árbol generador T todas las aristas que reporten como extremo a a, de acuerdo con el orden del abecedario. Luego:

$$T = \left\{ \left[a, \boxed{d}\right], [a, e], [a, g], [a, h] \right\} \tag{33}$$

Solución del ejemplo 6.16

Gráficamente:



Solución del ejemplo 6.16

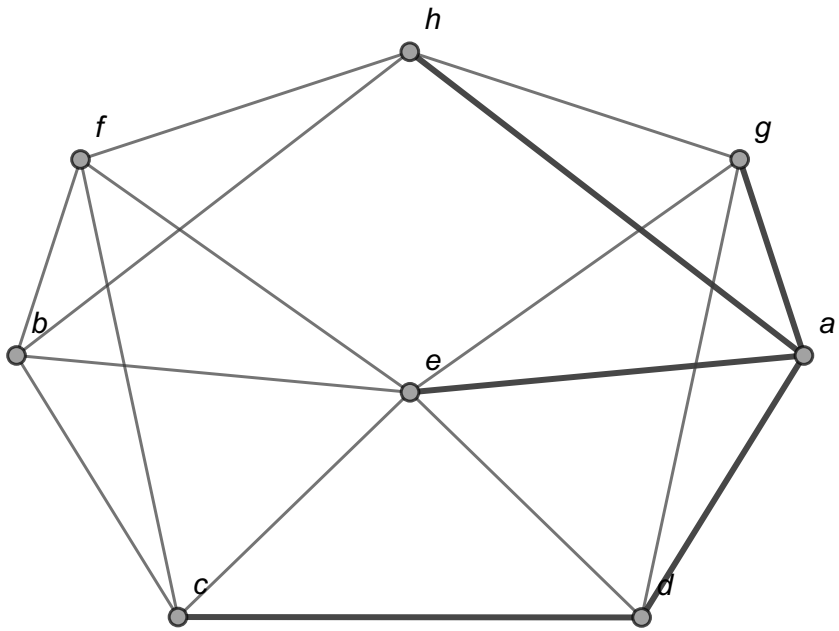
Los lados del árbol T se simbolizan con corchetes cuadrados ($[]$) en vista de que la definición 5.1 del capítulo anterior, estipula que se emplean ese tipo de paréntesis cuando se hace referencia a las aristas de un grafo no dirigido, como ocurre en este tipo de ejercicios.
Ahora, por el orden de los lados incluidos en T, se agregan a T todas las aristas incidentes a d que no formen un circuito. Esto se ha señalado en 33 con un recuadro en d. Se busca que no se generen circuitos pues como ya sabemos, un árbol no posee este tipo de trayectorias. Solo se añade el lado $[d, c]$ pues $[d, e]$ y $[d, g]$ generan rutas circuitales. Es fácil reconocer si una arista a agregar forma un circuito, pues si con los lados de T y la arista elegible se "cierra una figura" (un triángulo, un cuadrilátero, entre otras), significa en este contexto, tener un camino circuital.

Solución del ejemplo 6.16

El lado $[d, e]$ forma un circuito en este caso, pues cierra el triángulo Δdea y la arista $[d, g]$ también perfila un circuito pues cierra el triángulo Δdga. Luego:

$$T = \left\{ \left[a, \boxed{d}\right], [a, \boxed{e}], [a, g], [a, h], [d, c] \right\} \tag{34}$$

Visualmente:



Solución del ejemplo 6.16

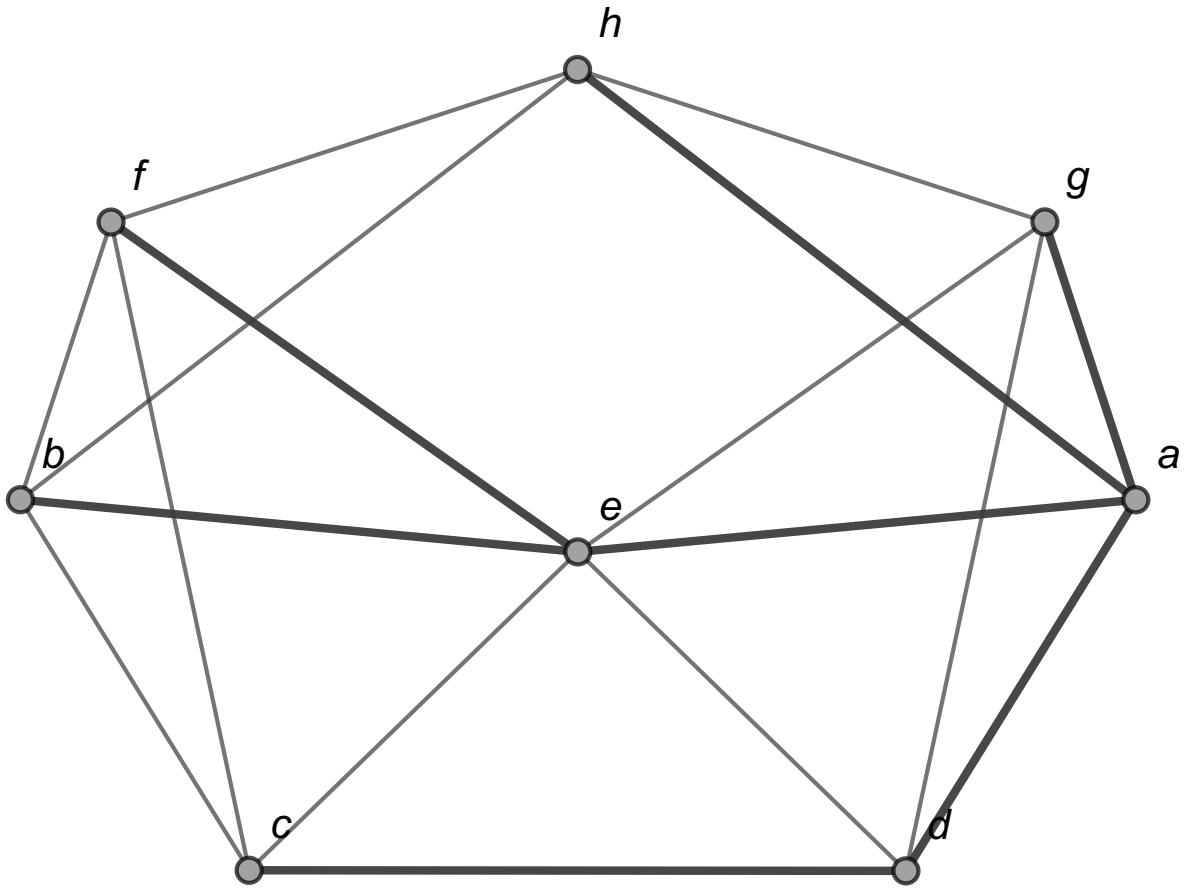
Corresponde ubicarnos en el vértice e como se aprecia en 34. Las aristas incidentes a e que no forman circuitos son $[e, b]$ y $[e, f]$:

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

El orden del abecedario interesa al incluir en T los lados $[e, b]$ y $[e, f]$, por lo que, si se pone primero $[e, f]$ y posteriormente $[e, b]$ se estaría incurriendo en un error. Se ha finalizado pues T posee 7 lados.

Solución del ejemplo 6.16

Gráficamente:



Nota

El algoritmo "buscar primero a lo ancho" recibe este nombre pues las visitas a lo "ancho" sobre los vértices de un grafo conexo, parten de un nodo raíz y se van agregando las aristas por niveles al árbol T. Esto hace que el recorrido sea literalmente una trayectoria a lo ancho, pues en primera instancia se añaden al árbol de expansión T, todos los nodos posibles ubicados en el nivel 1, luego, se agregan todos los vértices ubicados en el nivel 2 y así sucesivamente, hasta completar la totalidad de niveles. En este ejemplo el árbol a lo "ancho" solo contiene dos niveles, los nodos vinculados con el nivel 1 son d, e, g y h y, los vértices asociados con el nivel 2 corresponden a c, b y f.

Solución del ejemplo 6.16

Por otro lado, al recurrir al algoritmo "buscar primero a lo largo", se parte del nodo raíz a. En a se tienen cuatro aristas incidentes $[a, d]$, $[a, e]$, $[a, g]$ y $[a, h]$, donde se selecciona $[a, d]$ para agregar al árbol T, por el orden del abecedario, al comparar d, e, g y h. Al ubicarnos en el vértice d, se tienen varios lados elegibles $[d, c]$, $[d, e]$ y $[d, g]$, al comparar c, e y g, se escoge a $[d, c]$ por el orden del abecedario utilizado. Ahora, al tomar el vértice c hay varias aristas seleccionables $[c, b]$, $[c, e]$ y $[c, f]$, de ellas se elige a $[c, b]$. En b, los lados candidatos son $[b, e]$, $[b, f]$ y $[b, h]$, escogiendo a $[b, e]$ de acuerdo con el orden del abecedario, al comparar e, f y h.

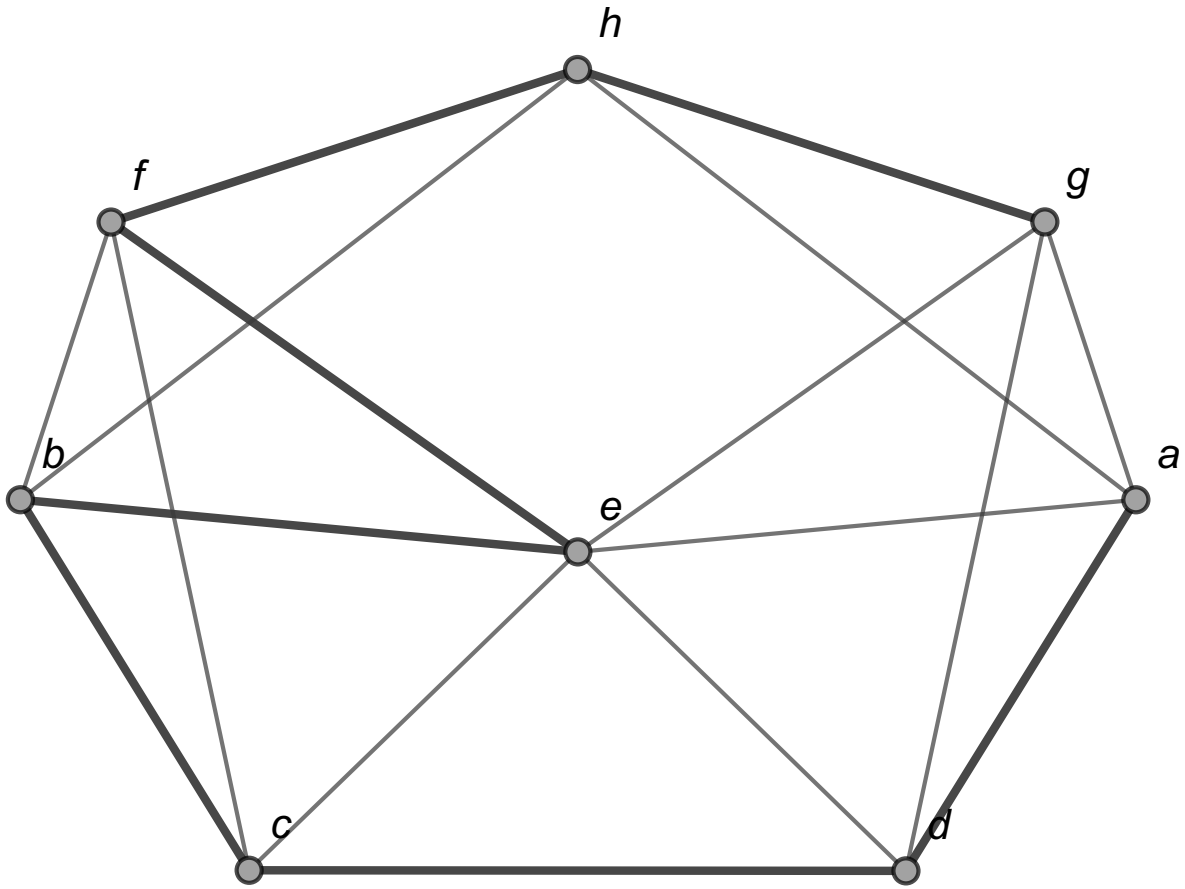
Solución del ejemplo 6.16

Luego, en e las aristas incidentes elegibles son en orden: $[e, a]$, $[e, c]$, $[e, d]$, $[e, f]$ y $[e, g]$, el lado $[e, a]$ no se puede seleccionar pues cierra un polígono $(e \to a \to d \to c \to b \to e)$, la arista $[e, d]$ cierra el cuadrilátero $\square edcb$ y el lado siguiente $[e, f]$ se escoge al no formar un circuito. En f, las aristas seleccionables son en el orden del abecedario: $[f, b]$, $[f, c]$ y $[f, h]$, $[f, b]$ cierra el triángulo Δfbe y $[f, c]$ genera el circuito $f \to c \to b \to e \to f$, por lo tanto, se selecciona la arista $[f, h]$. Luego en el nodo h, hay tres lados incidentes a escoger $[h, a]$, $[h, b]$ y $[h, g]$, los primeros dos forman circuitos en cuyo caso se elige $[h, g]$. Ya se tienen 7 aristas en T, se ha finalizado:

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

Solución del ejemplo 6.16

Gráficamente:



Solución del ejemplo 6.16

En el contexto del teorema 26, formalmente el proceso termina cuando el nodo w vuelve a ser el vértice de partida. En el procedimiento descrito con anterioridad, w quedó igual a g. El algoritmo **BPL** propone que se debe reasignar w al padre de g, es decir, $w = h$ y se analiza si se pueden añadir más aristas, como no es posible hacerlo (pues todas forman circuitos), se actualiza w como el padre de h, en consecuencia, $w = f$, se intenta allí agregar más aristas, al no ser posible, se reasigna w como el padre de f, por consiguiente, $w = e$ y así sucesivamente hasta que $w = a$, en ese momento se termina la ejecución del algoritmo. Normalmente por motivos de efectividad, no se detalla este recorrido hacia atrás, en caso de que T ya posea los $n - 1$ lados requeridos, pese a ello, el alumno no debe olvidar el criterio de finalización del algoritmo "buscar primero a profundidad".

Nota

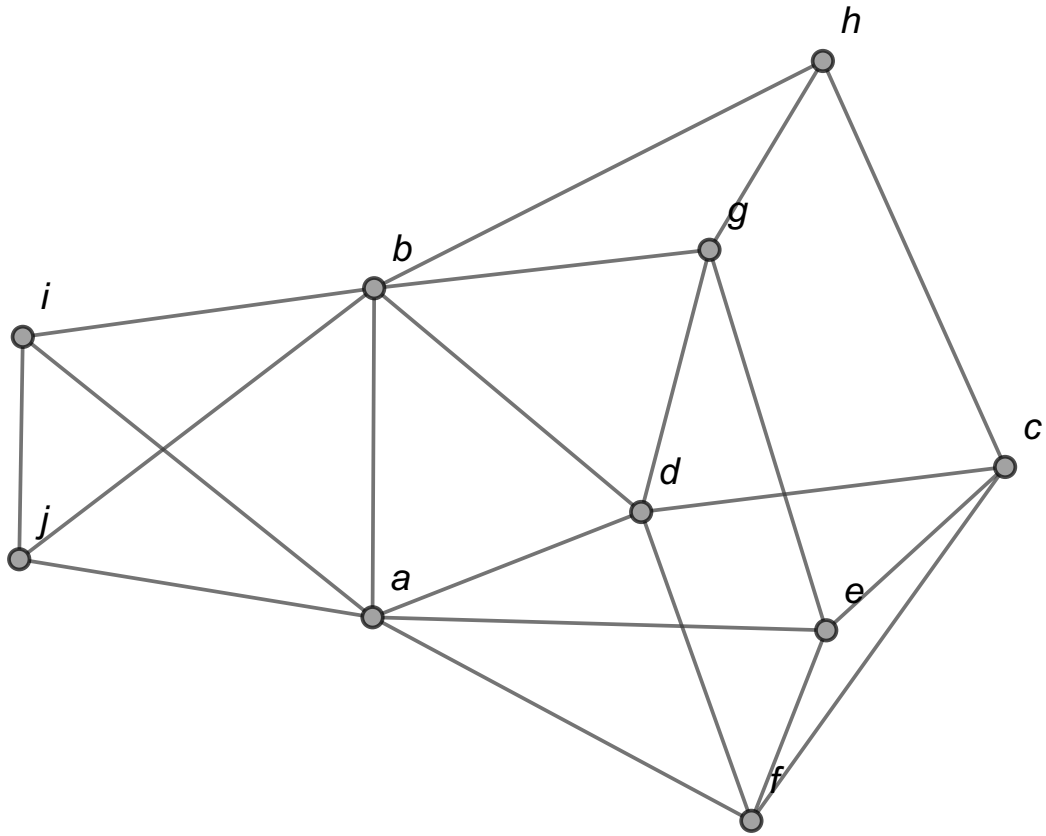
El algoritmo "buscar primero a lo largo" acoge esta denominación al describir una trayectoria en profundidad, partiendo de un vértice raíz. El método establece que al llegar al último nodo a lo largo, se debe regresar a su padre e incluir más aristas en caso de ser necesario y repetir esta lógica hasta volver al nodo raíz.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-152.zip

Example (6.17)

Halle un árbol de expansión a lo ancho y a lo largo del grafo dado, suponiendo el orden del abecedario para el conjunto de nodos V.



Solución del ejemplo 6.17

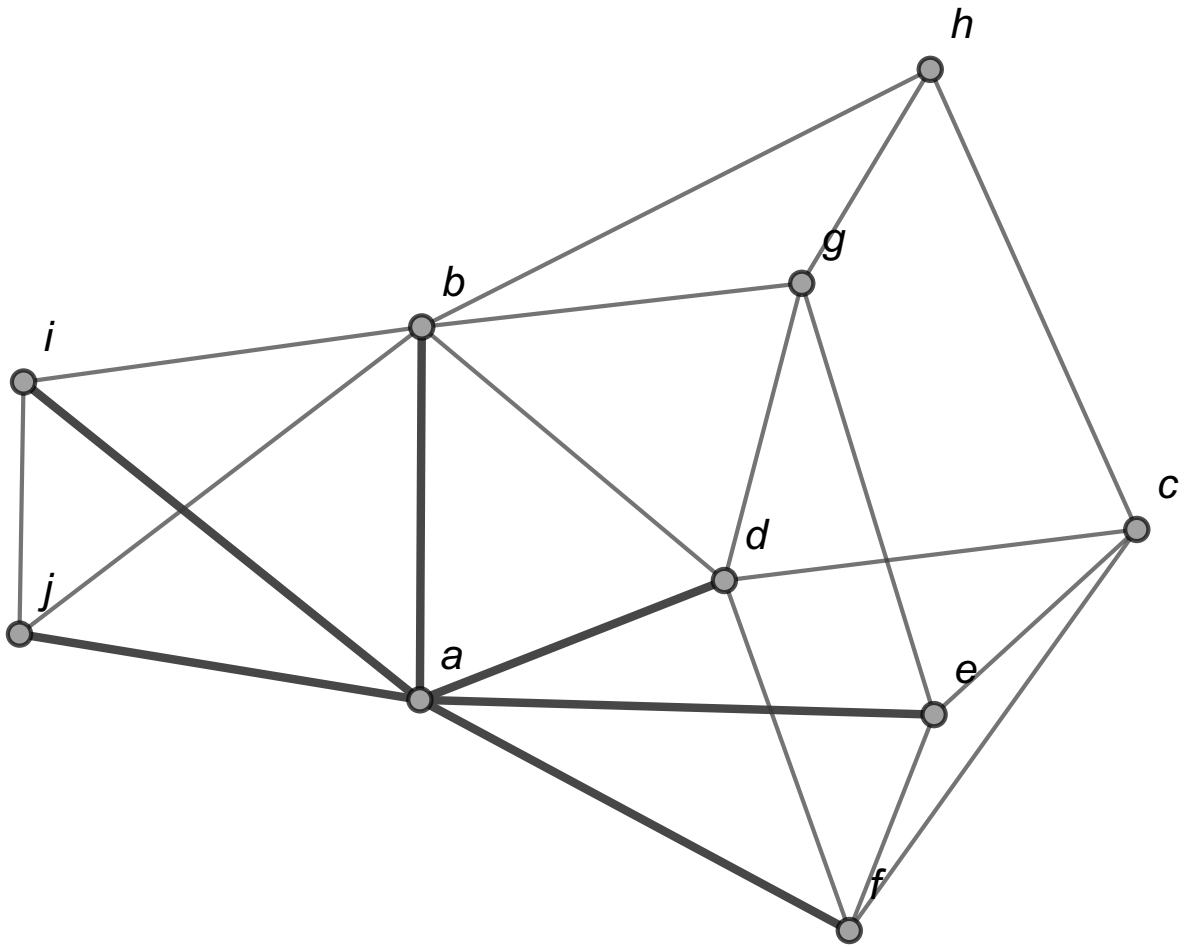
En este ejemplo, el orden a considerar en V corresponde a: $V = \{a, b, c, d, e, f, g, h, i, j\}$. Además, la cantidad de nodos en el grafo es $n = 10$, por lo que, **BPA** y **BPL** terminan cuando el árbol generador T a construir, tenga $n - 1 = 10 - 1 = 9$ lados.

En el algoritmo **BPA** se parte del vértice a como raíz del árbol de expansión T. Se tienen varias aristas incidentes a a, a saber: $[a, b]$, $[a, d]$, $[a, e]$, $[a, f]$, $[a, i]$ y $[a, j]$, todas se añaden a T :

$$T = \left\{ \left[a, \boxed{b}\right], [a, d], [a, e], [a, f], [a, i], [a, j] \right\} \tag{35}$$

Solución del ejemplo 6.17

Gráficamente:



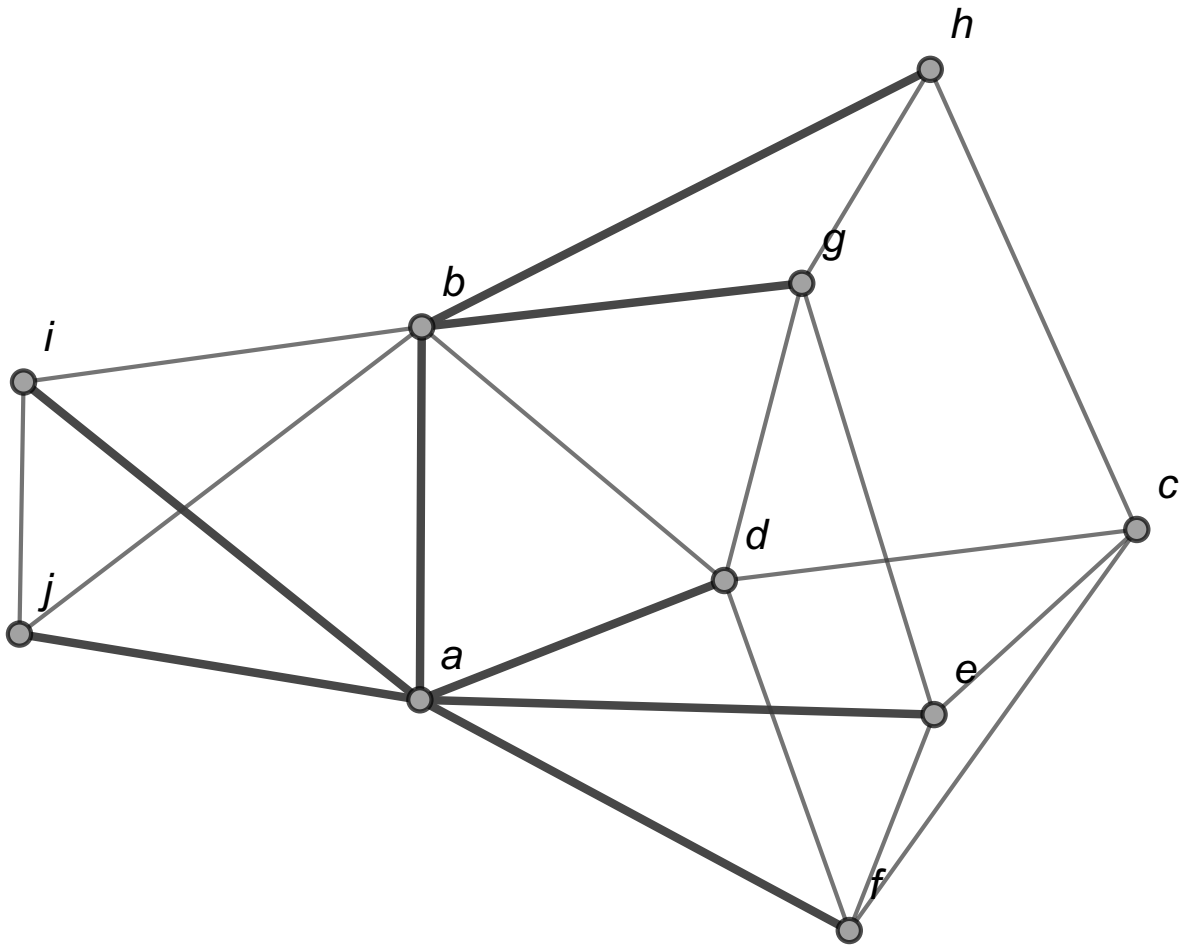
Solución del ejemplo 6.17

Ahora, al tomar el vértice b señalado en 35, los lados con extremo b seleccionables en el grafo son $[b,d]$, $[b,g]$, $[b,h]$, $[b,i]$ y $[b,j]$. La arista $[b,d]$ cierra el triángulo Δbda, por lo que, no se puede escoger. Los lados $[b,g]$ y $[b,h]$ sí son elegibles y al ser añadidos a T, las aristas $[b,i]$ y $[b,j]$ se descartan al formar rutas circuitales. Luego:

$$T = \left\{ \left[a, \boxed{b}\right], \left[a, \boxed{d}\right], [a,e], [a,f], [a,i], [a,j], [b,g], [b,h] \right\} \qquad (36)$$

Solución del ejemplo 6.17

Visualmente:



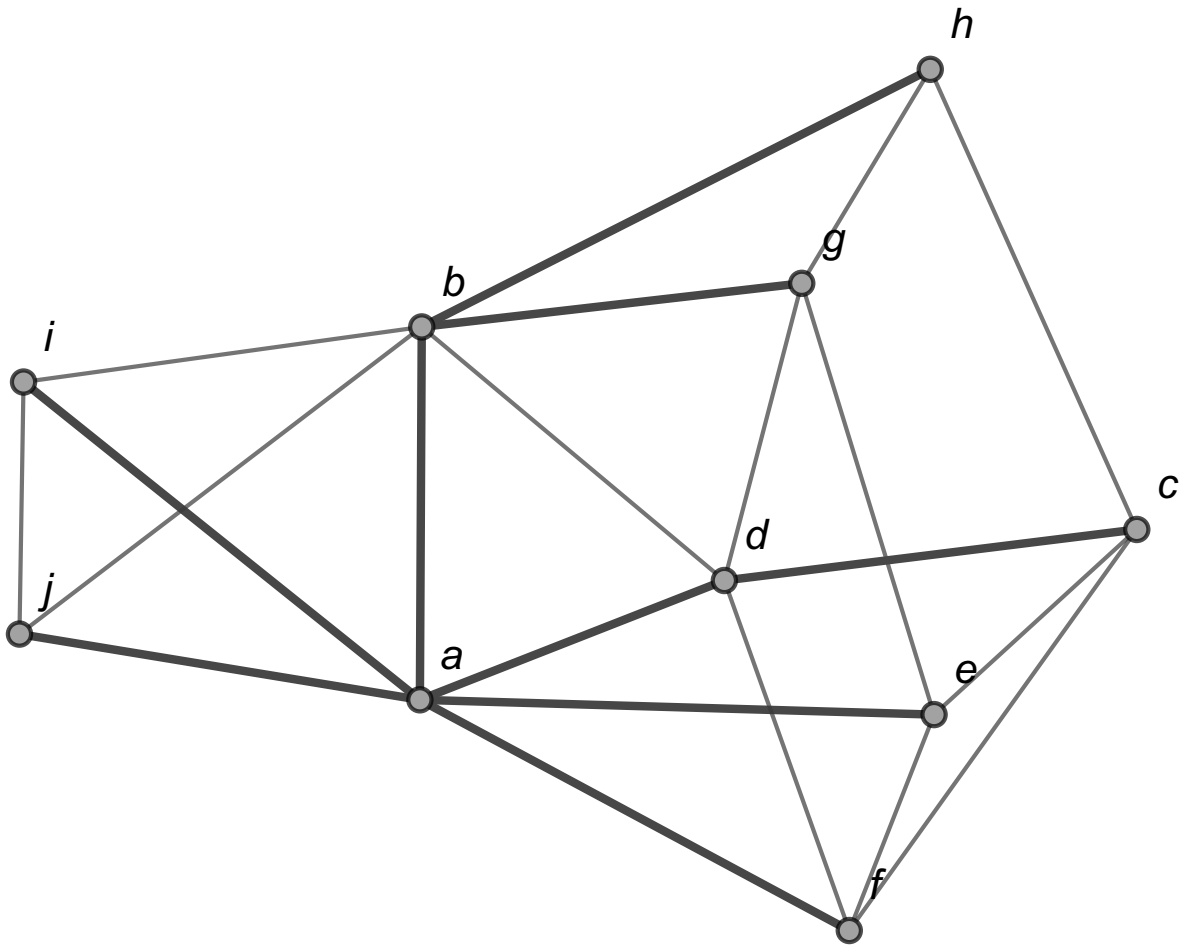
Solución del ejemplo 6.17

En 36, se aprecia que el nodo siguiente es d. Las aristas seleccionables con extremo d son $[d, b]$, $[d, c]$, $[d, f]$ y $[d, g]$, de todas ellas solo $[d, c]$ no forma un circuito entonces se añade al árbol T. En este punto, T posee 9 lados, se ha finalizado:

$$T = \{[a, b], [a, d], [a, e], [a, f], [a, i], [a, j], [b, g], [b, h], [d, c]\} \quad (37)$$

Solución del ejemplo 6.17

Gráficamente:



Solución del ejemplo 6.17

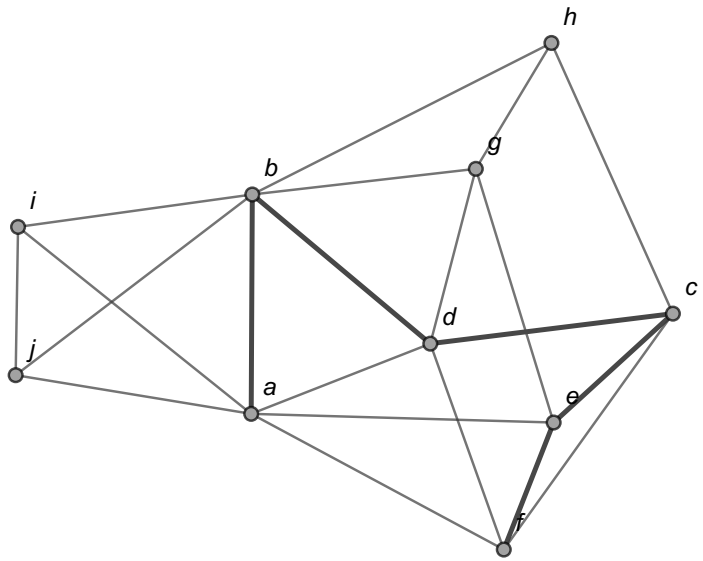
Por otra parte, al utilizar el algoritmo **BPL** se parte del nodo a. En a, hay varias aristas incidentes: $[a, b]$, $[a, d]$, $[a, e]$, $[a, f]$, $[a, i]$ y $[a, j]$, se selecciona $[a, b]$ por el orden del abecedario, al comparar b, d, e, f, i y j. Ahora, en b los lados a considerar son $[b, d]$, $[b, g]$, $[b, h]$, $[b, i]$ y $[b, j]$, se elige $[b, d]$ por el orden del alfabeto, al comparar d, g, h, i y j. Ahora, ubicados en el vértice d, las aristas incidentes a d elegibles son $[d, a]$, $[d, c]$, $[d, f]$ y $[d, g]$, se escoge $[d, c]$ pues $[d, a]$ genera un circuito. Los lados seleccionables con extremo c son $[c, e]$, $[c, f]$ y $[c, h]$, se elije $[c, e]$. Luego, en e las aristas candidatas son $[e, a]$, $[e, f]$ y $[e, g]$, de ellas $[e, a]$ no se agrega a T al generar el circuito $e \to a \to b \to d \to c \to e$, por consiguiente, se escoge a $[e, f]$. En el nodo f se ha alcanzado el mayor nivel de profundidad del árbol T, pues los lados seleccionables $[f, a]$, $[f, c]$ y $[f, d]$ forman circuitos.

Solución del ejemplo 6.17

Se observa que no se ha finalizado el proceso pues T no contiene 9 aristas, en este momento T posee únicamente 5 lados:

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

Visualmente:



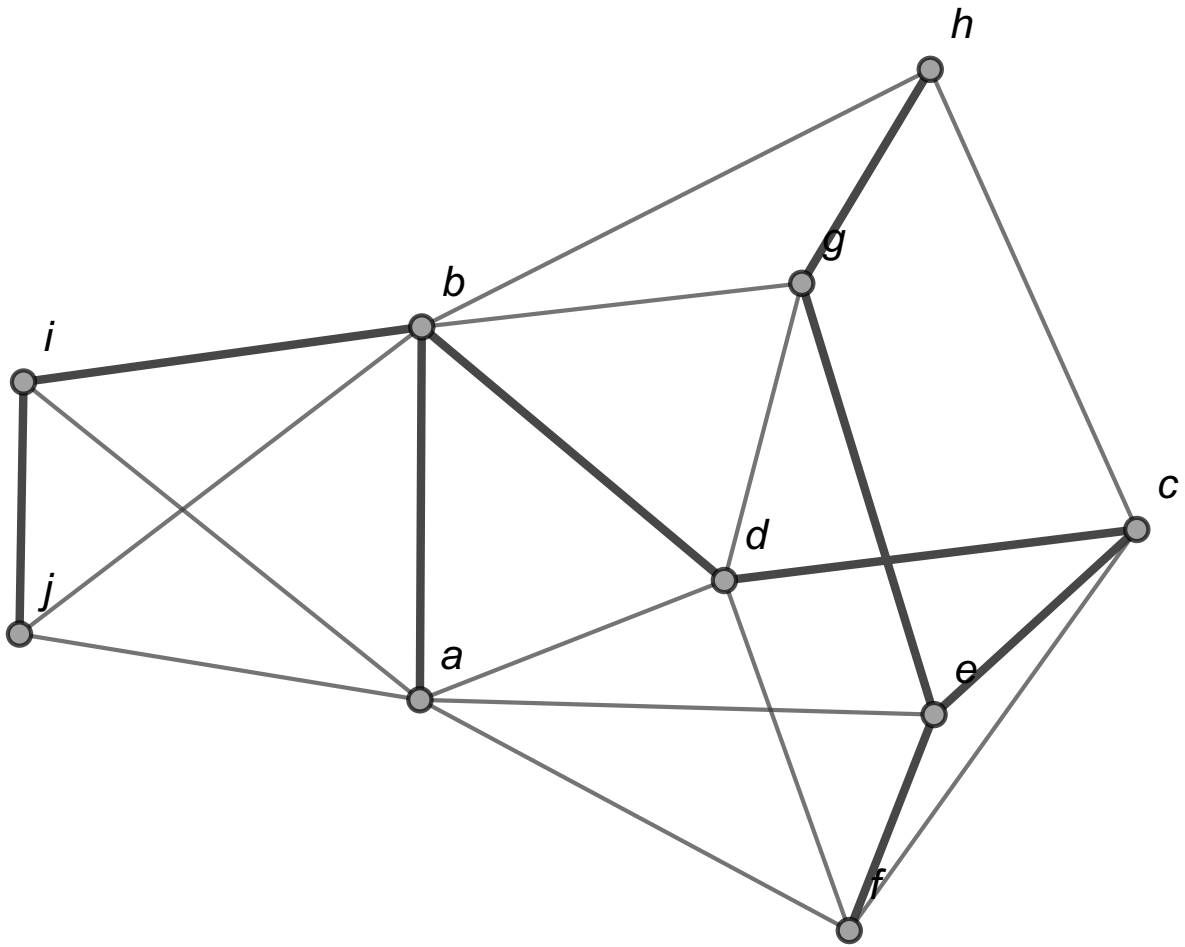
Solución del ejemplo 6.17

El teorema 26 propone ahora, un recorrido hacia atrás. Se regresa al padre de f, en e y se agrega una arista de ser posible. El lado $[e, g]$ no forma un circuito, se añade a T. Ubicados en el nodo g, se incluye $[g, h]$ al árbol T dado que las otras aristas incidentes elegibles derivan en rutas circuitales. En h todos los lados incidentes a seleccionar forman un circuito, se regresa al padre de h en g, al padre de g en e, a su padre en c, al padre de c en d, a su padre en b y hasta ese momento las aristas candidatas en cada punto generaron circuitos, en b se determina un lado que no lo hace, la arista $[b, i]$ que se añade a T y en el vértice i el lado $[i, j]$ se incluye también, al no perfilar un circuito. Se ha finalizado pues T es de cardinalidad 9 :

$$T = \{[a, b], [b, d], [d, c], [c, e], [e, f], [e, g], [g, h], [b, i], [i, j]\} \quad (38)$$

Solución del ejemplo 6.17

Gráficamente:



Nota

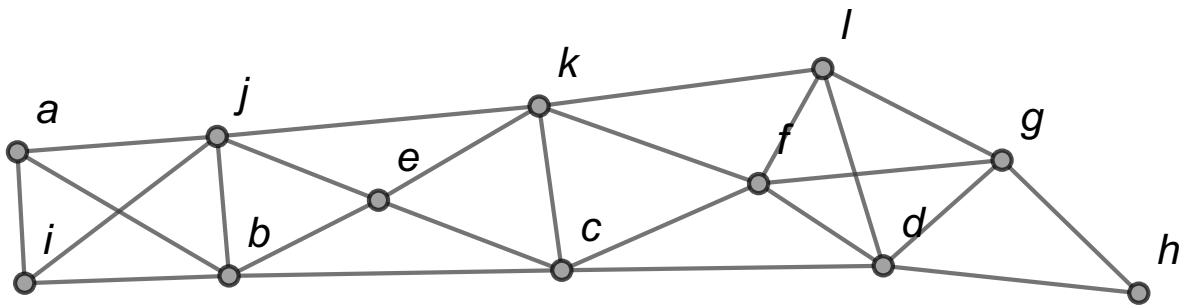
Se reitera al estudiante que los dibujos de los árboles obtenidos a lo "ancho" y a lo "largo" son importantes para efectos de explicación, pero irrelevantes con el objetivo de dar respuesta al ejercicio. La solución del ejemplo está dada, no por la figura de los árboles en cuestión, sino por los conjuntos expuestos en 37 y 38.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-153.zip

Example (6.18)

Determine en el grafo compartido un árbol recubridor mediante los métodos **BPA** y **BPL**, con el orden en el conjunto V de vértices, $V = \{k, i, l, j, a, c, h, f, d, b, e, g\}$.



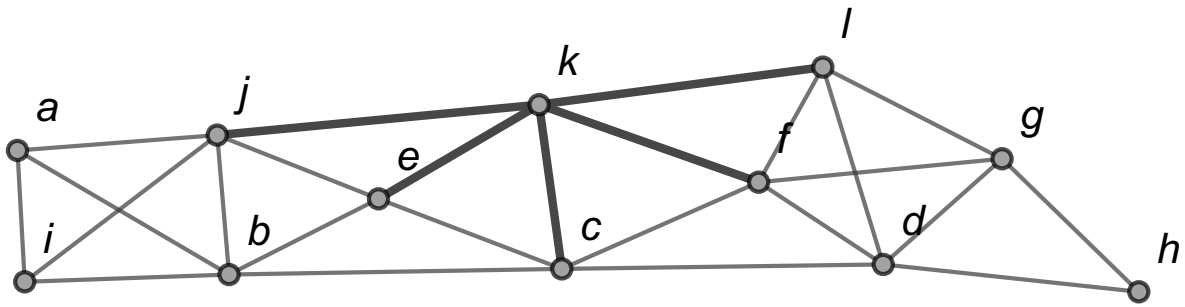
Solución del ejemplo 6.18

El número n de nodos del grafo es $n = 12$. **BPA** y **BPL** finalizan cuando el árbol de expansión T, contenga $n - 1 = 12 - 1 = 11$ lados.
En el método **BPA** se toma como raíz de T, el vértice k, esto de acuerdo con el orden dado de V. Las aristas incidentes a k se añaden al árbol T respetando el orden definido para V. Estos lados son $[k, c]$, $[k, e]$, $[k, f]$, $[k, j]$ y $[k, l]$ donde en V se tiene:
$V = \left\{k, i, \boxed{l}, \boxed{j}, a, \boxed{c}, h, \boxed{f}, d, b, \boxed{e}, g\right\}$. Lo anterior significa que las aristas se deben agregar a T en el orden: $[k, l]$, $[k, j]$, $[k, c]$, $[k, f]$ y $[k, e]$.
Por consiguiente:

$$T = \left\{\left[k, \boxed{l}\right], [k, j], [k, c], [k, f], [k, e]\right\}$$

Solución del ejemplo 6.18

Visualmente:



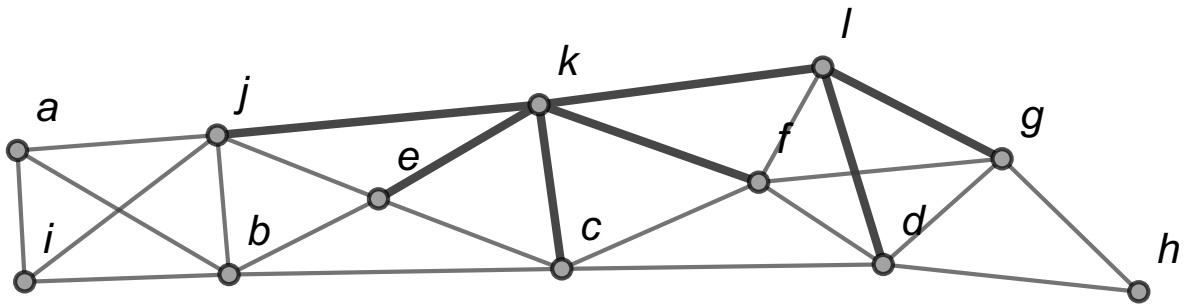
Solución del ejemplo 6.18

Al tomar el nodo l, los lados incidentes seleccionables son $[l, d]$, $[l, f]$ y $[l, g]$. La arista $[l, f]$ cierra el triángulo Δlfk, por lo que se descarta. Se añaden a T los lados $[l, d]$ y $[l, g]$ donde:
$V = \left\{k, i, l, j, a, c, h, f, \boxed{d}, b, e, \boxed{g}\right\}$. Como consecuencia, estas aristas se colocan en T en el orden $[l, d]$ y $[l, g]$:

$$T = \left\{ \left[k, \boxed{l}\right] , \left[k, \boxed{j}\right] , [k, c] , [k, f] , [k, e] , [l, d] , [l, g] \right\}$$

Solución del ejemplo 6.18

Gráficamente:



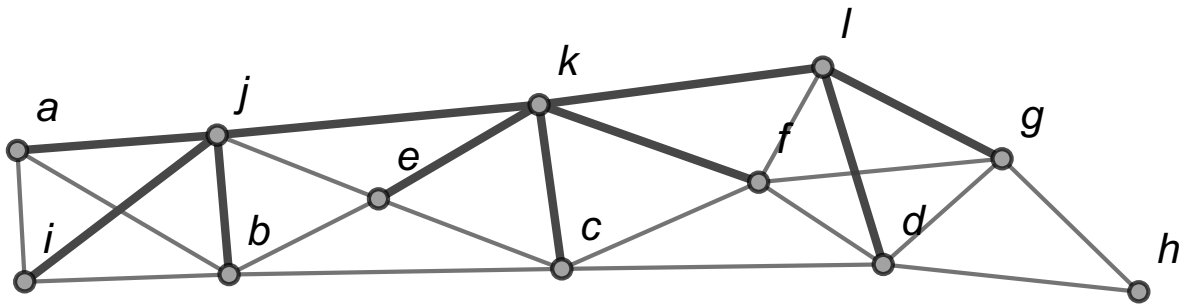
Solución del ejemplo 6.18

Ahora, en j los lados elegibles son $[j,a]$, $[j,b]$, $[j,e]$ y $[j,i]$. De ellos, se excluye $[j,e]$ al formar un circuito, se agregan al árbol T las aristas $[j,a]$, $[j,b]$ y $[j,i]$ con $V = \left\{k, \boxed{i}, l, j, \boxed{a}, c, h, f, d, \boxed{b}, e, g\right\}$, induciendo el orden $[j,i]$, $[j,a]$ y $[j,b]$:

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

Solución del ejemplo 6.18

Visualmente:



Solución del ejemplo 6.18

En c, todos los lados seleccionables generan rutas circuitales. Se toma entonces el nodo f :

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

Del mismo modo, todas las aristas elegibles con extremo f forman circuitos. Se pasa al vértice e :

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

De manera análoga, todos los lados a escoger incidentes a e generan circuitos. Se toma el nodo d :

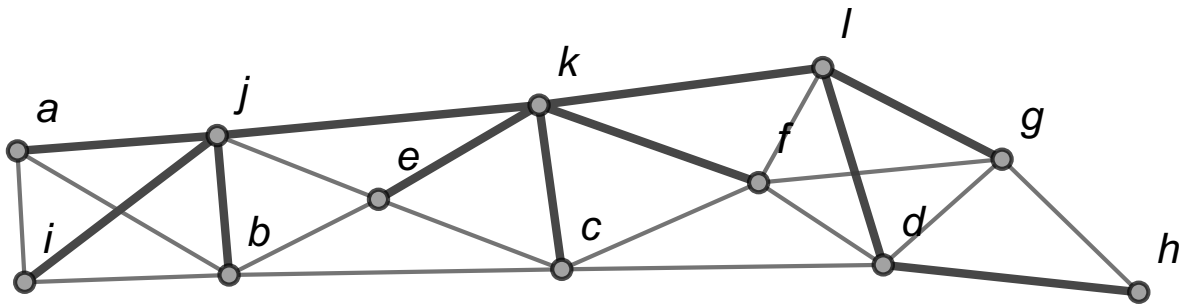
$$T = \left\{ \left[k, \boxed{l}\right], \left[k, \boxed{j}\right], \left[k, \boxed{c}\right], \left[k, \boxed{f}\right], \left[k, \boxed{e}\right], \left[l, \boxed{d}\right], [l, g], [j, i], [j, a], [j, b] \right\}$$

Solución del ejemplo 6.18

En d, se descartan las aristas $[d, c]$, $[d, f]$ y $[d, g]$ al formar circuitos, solo es posible añadir a T el lado $[d, h]$:

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

Se ha finalizado pues T contiene 11 aristas. Gráficamente:



Solución del ejemplo 6.18

Por otra parte, el procedimiento **BPL** parte también del vértice k. Al considerar las aristas incidentes a k: $[k, c]$, $[k, e]$, $[k, f]$, $[k, j]$ y $[k, l]$ y, el orden de $V = \left\{k, i, \boxed{l}, \boxed{j}, a, \boxed{c}, h, \boxed{f}, d, b, \boxed{e}, g\right\}$, se aprecia que el primer lado de T es $[k, l]$. En l, se presentan las aristas elegibles $[l, d]$, $[l, f]$ y $[l, g]$ donde $V = \left\{k, i, l, j, a, c, h, \boxed{f}, \boxed{d}, b, e, \boxed{g}\right\}$, en cuyo caso, se escoge $[l, f]$. En f, los lados seleccionables son $[f, c]$, $[f, d]$, $[f, k]$ y $[f, g]$ y, $V = \left\{\boxed{k}, i, l, j, a, \boxed{c}, h, f, \boxed{d}, b, e, \boxed{g}\right\}$. De acuerdo con el orden de V, corresponde escoger $[f, k]$, sin embargo, $[f, k]$ genera un circuito. La arista que prosigue en el orden de V es $[f, c]$, ésta se agrega a T. Ahora en el nodo c, los lados a escoger son $[c, b]$, $[c, d]$, $[c, e]$ y $[c, k]$, en donde $V = \left\{\boxed{k}, i, l, j, a, c, h, f, \boxed{d}, \boxed{b}, \boxed{e}, g\right\}$, a razón de ello, se debería seleccionar $[c, k]$ pero esta arista cierra el cuadrilátero $\square cklf$.

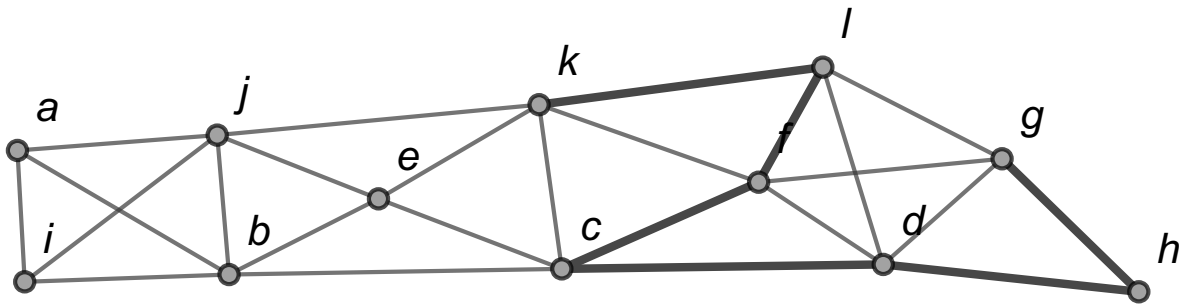
Solución del ejemplo 6.18

El lado que continua en el orden de V es $[c,d]$ y al no formar un circuito se añade a T. En el vértice d, las aristas elegibles son $[d,f]$, $[d,g]$, $[d,h]$ y $[d,l]$ con $V=\left\{k,i,\boxed{l},j,a,c,\boxed{h},\boxed{f},d,b,e,\boxed{g}\right\}$. En función del orden de V, el lado a agregar corresponde a $[d,l]$, sin embargo, forma un circuito, por lo que, se pasa a la arista $[d,h]$ según el orden de V, como ésta no perfila un circuito se añade a T. Ahora, se agrega a T el único lado disponible con extremo h: $[h,g]$. En g, no se puede agregar ninguna arista pues de las posibles candidatas todas forman rutas circuitales. Esto indica que se ha alcanzado el mayor nivel de profundidad en el árbol T y no se ha finalizado pues en este momento T cuenta con 6 lados y no con 9 :

$$T=\{[k,l],[l,f],[f,c],[c,d],[d,h],[h,g]\}$$

Solución del ejemplo 6.18

Gráficamente:



Solución del ejemplo 6.18

El proceso prosigue tomando al padre de g, el nodo h y analizando si se puede añadir en h un nuevo lado para T. Esto no es viable, por lo que, se regresa al padre de h en d. En el vértice d tampoco es factible incluir una nueva arista al árbol T. Se regresa al padre de d, en c y allí sí se identifican lados seleccionables: $[c, b]$, $[c, e]$ y $[c, k]$. Como $V = \left\{\boxed{k}, i, l, j, a, c, h, f, d, \boxed{b}, \boxed{e}, g\right\}$ se aprecia que la nueva arista a agregar a T es $[c, b]$ pues $[c, k]$ genera un circuito. Ubicados en el vértice b, hay varias posibilidades de elección $[b, a]$, $[b, e]$, $[b, i]$ y $[b, j]$ con $V = \left\{k, \boxed{i}, l, \boxed{j}, \boxed{a}, c, h, f, d, b, \boxed{e}, g\right\}$, corresponde entonces incluir en T el lado $[b, i]$. En i, las aristas seleccionables son $[i, a]$ y $[i, j]$ donde $V = \left\{k, i, l, \boxed{j}, \boxed{a}, c, h, f, d, b, e, g\right\}$, dado que $[i, j]$ no produce un circuito se añade a T.

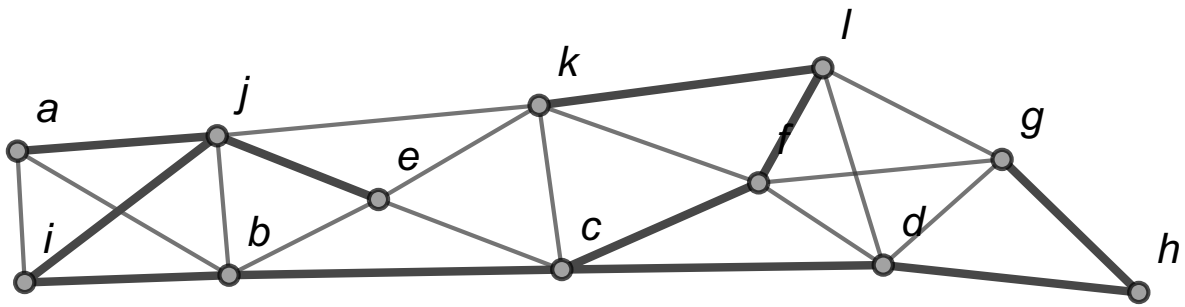
Solución del ejemplo 6.18

En j, se tienen los lados a escoger $[j,a]$, $[j,b]$, $[j,e]$ y $[j,k]$ con $V = \left\{\boxed{k}, i, l, j, \boxed{a}, c, h, f, d, \boxed{b}, \boxed{e}, g\right\}$, por lo tanto, se incluye en T la arista $[j,a]$, al descartarse $[j,k]$ por la ocurrencia de un circuito. Falta aún, un lado más a agregar en T para finalizar. En a, el lado $[a,i]$ cierra el triángulo Δaij y la arista $[a,b]$ perfila otro circuito, a razón de ello, se debe regresar al padre de a, el nodo j. En j, la única arista que no forma un circuito es $[j,e]$, añadiéndose al árbol T. Luego:

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

Solución del ejemplo 6.18

Visualmente:

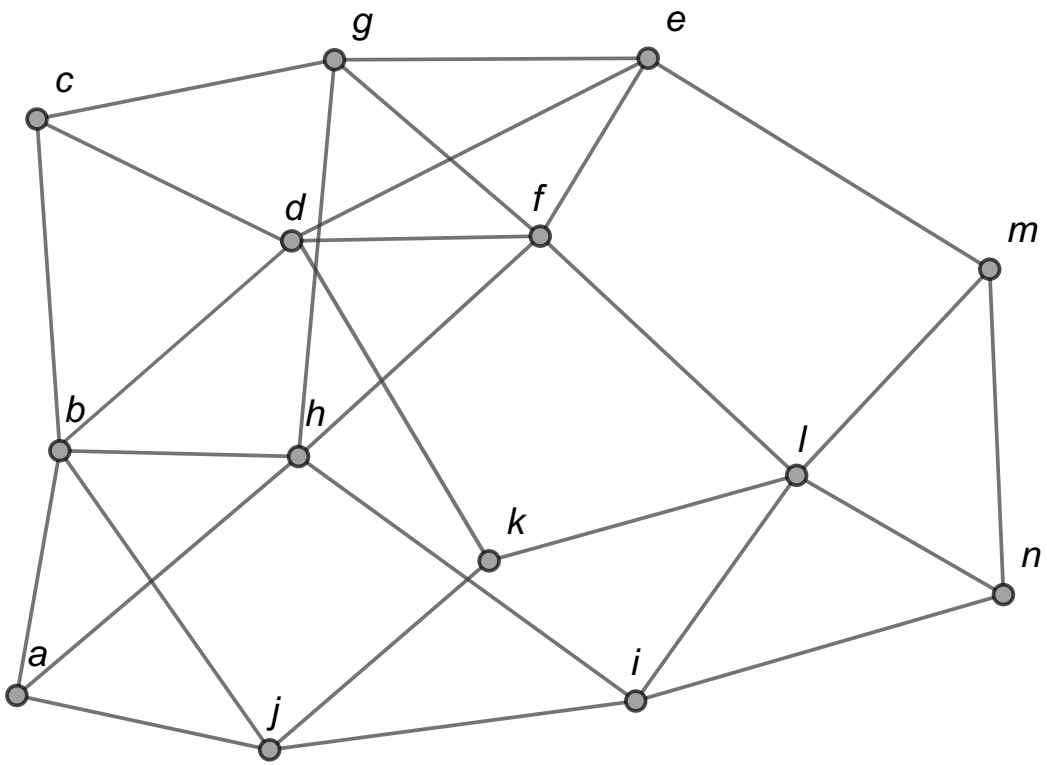


Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-154.zip

Example (6.19)

Encuentre con apoyo de software, un árbol generador del grafo conexo presentado a continuación, usando los algoritmos **BPA** y **BPL**, con $V = \{m, g, d, k, j, a, b, l, i, h, e, n, c, f\}$.



Solución del ejemplo 6.19

El paquete **VilCretas** incorpora dos comandos de importancia para devolver en el software *Mathematica* un árbol generador a lo "ancho" y a lo "largo", las instrucciones son: `BuscarPrimeroAncho` y `BuscarPrimeroLargo`, respectivamente. Ambas, reciben como argumento el grafo conexo sobre el cual se desea aplicar los algoritmos **BPA** y **BPL**. Las dos sentencias integran las opciones `orden` y `animacion -> True`. El atributo `orden` especifica el orden a emplear en los nodos del grafo cuando se utilizan los métodos "buscar primero a lo ancho" y "buscar primero a lo largo". Si se omite, *Wolfram* selecciona por cuenta propia un orden para los vértices asociado al orden empleado en las aristas cuando el grafo fue creado. La opción `animacion` muestra en una animación, paso a paso y en orden, cada uno de los lados del árbol de expansión a lo "ancho", o bien, a lo "largo", según corresponda.

Solución del ejemplo 6.19

En este ejemplo, al recurrir a los comandos BuscarPrimeroAncho y BuscarPrimeroLargo, se obtiene:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, h}, {b, h}, {b, j}, {b, c},
{b, d}, {c, d}, {d, k}, {d, f}, {d, e}, {e, f}, {f, g},
{f, h}, {f, l}, {g, h}, {h, i}, {i, j}, {i, l}, {j, k},
{k, l}, {e, g}, {g, c}, {a, j}, {l, m}, {e, m}, {n, m},
{l, n}, {n, i}}];
BuscarPrimeroAncho[grafo, orden -> {m, g, d, k, j, a, b, l,
i, h, e, n, c, f}]
BuscarPrimeroLargo[grafo, orden -> {m, g, d, k, j, a, b, l,
i, h, e, n, c, f}]
```

Solución del ejemplo 6.19

Se obtiene la siguiente salida:

Out[ ] =

{{m, l}, {m, e}, {m, n}, {k, l}, {l, i}, {l, f}, {g, e}, {d, e}, {k, j}, {i, h}, {g, c}, {d, b}, {j, a}}
{{m, l}, {k, l}, {d, k}, {d, b}, {j, b}, {j, a}, {a, h}, {g, h}, {g, e}, {e, f}, {g, c}, {i, h}, {i, n}}

Nota

En el **Out[ ]** algunas aristas presentan un orden inverso en los vértices al compararse con la resolución manual, pese a ello, esto no genera ningún conflicto de interpretación pues el grafo es no dirigido. Por ejemplo, en la búsqueda a lo largo, el lado $[l, k]$ del árbol generador en un proceso manual, queda en ese orden, sin embargo, en el **Out[ ]** anterior, *Wolfram* retornó {k, l}.

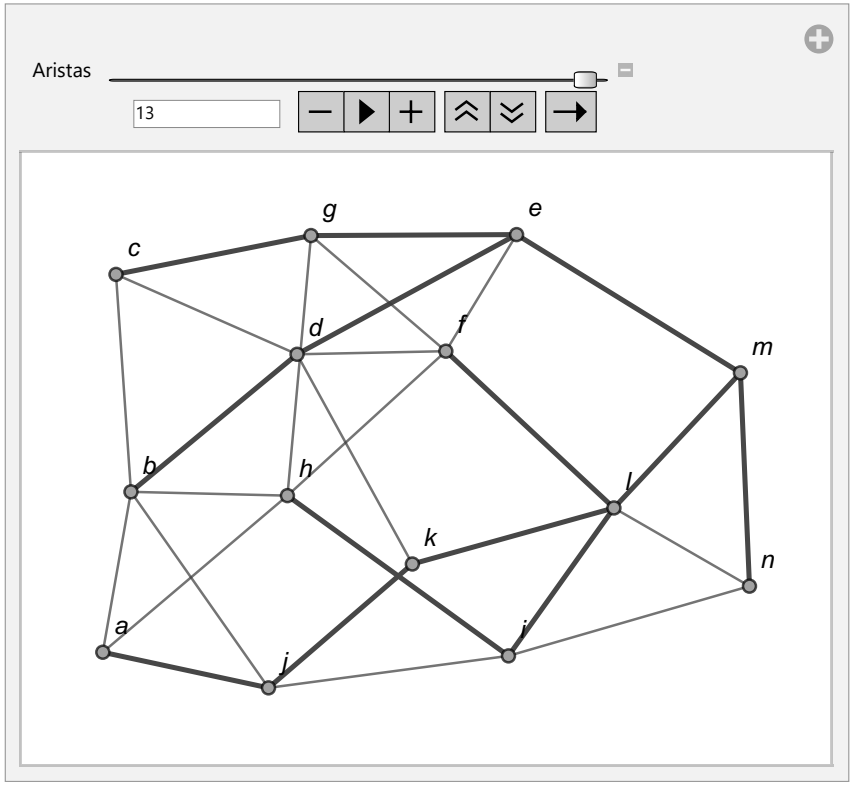
Solución del ejemplo 6.19

En la figura 5 se comparten las animaciones ofrecidas por el software al correr el siguiente **In[ ]**, donde se dispone de la opción `animacion -> True`:

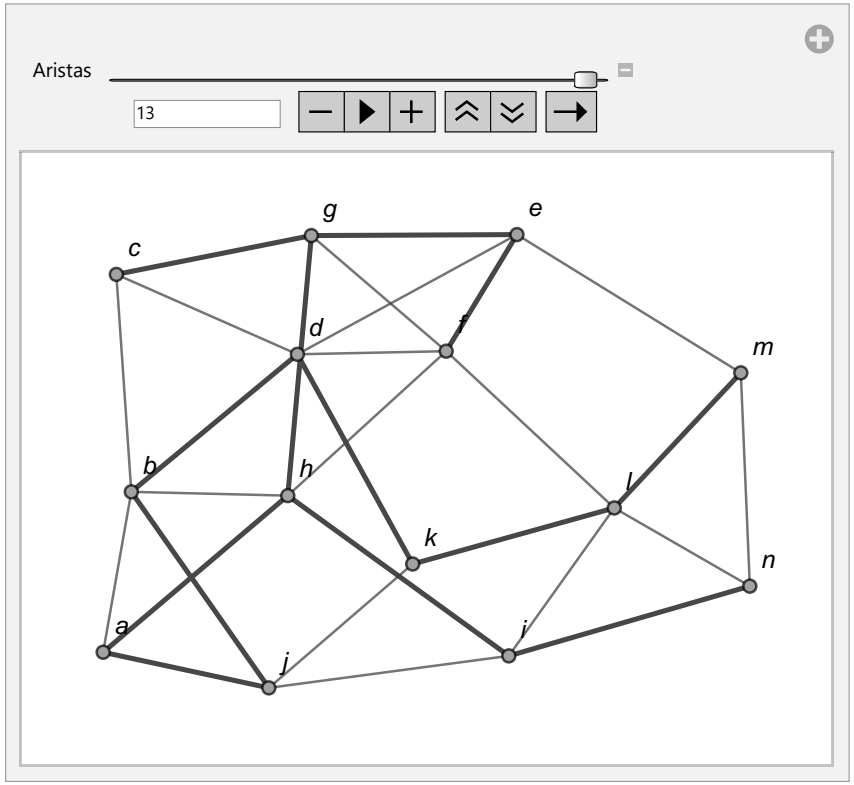
In[ ] :=

```
BuscarPrimeroAncho[grafo, orden -> {m, g, d, k, j, a, b, l,
i, h, e, n, c, f}, animacion -> True]
BuscarPrimeroLargo[grafo, orden -> {m, g, d, k, j, a, b, l,
i, h, e, n, c, f}, animacion -> True]
```

Este código también, retorna las listas de los lados del árbol recubridor a lo "ancho" y a lo "largo", respectivamente.



(a) Algoritmo "buscar primero a lo ancho"



(b) Algoritmo "buscar primero a lo largo"

Figura: Uso del atributo `animacion -> True`

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-155.zip

Utilización de la sentencia `BuscarPrimeroAncho`

Explicación en video

https://youtu.be/h7-cE7ixDFw

Empleo de la instrucción BuscarPrimeroLargo

Explicación en video

https://youtu.be/z3AtmeasOKM

- Cuando un grafo G conexo es ponderado, los árboles generadores que se obtienen de él, pueden contener un peso mínimo, es decir, la suma de los valores vinculados con las aristas que conforman el árbol recubridor, da como resultado un número mínimo. Existen dos importantes métodos para encontrar árboles de expansión minimales, éstos serán formalizados en la siguiente sección.

Árboles de expansión mínima

Prof. Enrique Vílchez Quesada

Universidad Nacional de Costa Rica

Definición 6.4

El concepto de árbol de expansión mínimal se enuncia inicialmente.

Definition (6.4)

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 se cuenta con dos algoritmos capaces de encontrar un árbol de expansión mínima en un grafo conexo con pesos. Estos métodos se denominan: "algoritmo de *Prim*" y "algoritmo de *Kruskal*". Veamos en qué consisten.

Teorema 6.8. Algoritmo de *Prim*

Theorem (6.8)

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

1. *Considere a T un árbol que contiene a v_1 y ninguna arista.*
2. *Si T posee $n-1$ elementos de E, se ha finalizado, T es un árbol de expansión mínima y 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 ésta sea incidente a alguno de los vértices de T y no forme un circuito en T. Si existe más de un lado $[v_i, v_j]$ de peso mínimo, elija el i menor, sino, el j más pequeño. Vaya al paso 2.*

Teorema 6.9. Algoritmo de *Kruskal*

Theorem (6.9)

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

1. *Considere a T un árbol que contiene todos los nodos de G y ningún lado.*
2. *Si T posee $n-1$ elementos de E, se ha finalizado, T es un árbol de expansión mínima y 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 no forme un circuito en T. Si existe más de uno elija cualquier arista. Vaya al paso 2.*

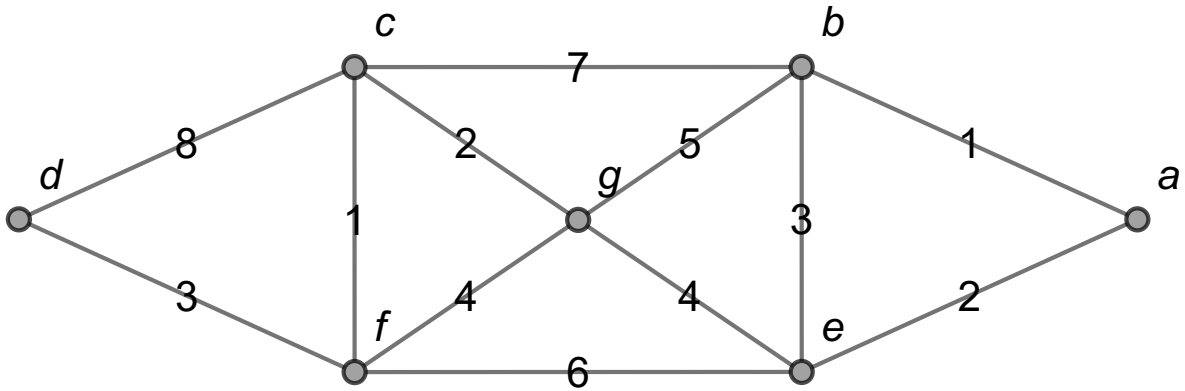
Comentario sobre los teoremas 32 y 33

El algoritmo de *Kruskal* proporciona una metodología de trabajo más abierta o flexible en comparación con el algoritmo de *Prim*. Si existen varias aristas con el mismo peso mínimo, *Kruskal* agrega cualquiera de estos lados al árbol T, siempre y cuando no se forme un circuito en T. Esto da la posibilidad de obtener árboles de expansión mínima distintos por medio de este procedimiento. El algoritmo de *Prim* a diferencia del método de *Kruskal*, devolverá un árbol generador mínimal único sobre un grafo conexo y ponderado, a partir de una lista fija y ordena de vértices V. También, se debe notar que los métodos de *Prim* y de *Kruskal* comparten con los algoritmos "buscar primero a lo ancho" y "buscar primero a lo largo", el hecho de requerir un orden para el conjunto de vértices V del grafo.

Se desarrollarán algunos ejemplos de uso sobre los procedimientos de *Prim* y de *Kruskal*.

Example (6.20)

En el grafo dado, determine un árbol generador de peso mínimo a través de los algoritmos de *Prim* y de *Kruskal*, suponiendo el orden del abecedario en el conjunto de nodos V.



Solución del ejemplo 6.20

En el presente ejercicio la cantidad n de vértices del grafo es $n = 7$, en consecuencia, *Prim* y *Kruskal* finalizan cuando el árbol T a construir, contenga un número de lados igual a $n - 1 = 7 - 1 = 6$. También, al observar el grafo se infiere la siguiente tabla de pesos:

Arista	Peso
$a \bullet\!\!-\!\!\bullet\ b$	1
$a \bullet\!\!-\!\!\bullet\ e$	2
$b \bullet\!\!-\!\!\bullet\ c$	7
$b \bullet\!\!-\!\!\bullet\ e$	3
$b \bullet\!\!-\!\!\bullet\ g$	5
$c \bullet\!\!-\!\!\bullet\ d$	8

Arista	Peso
$c \bullet\!\!-\!\!\bullet\ f$	1
$c \bullet\!\!-\!\!\bullet\ g$	2
$d \bullet\!\!-\!\!\bullet\ f$	3
$e \bullet\!\!-\!\!\bullet\ f$	6
$e \bullet\!\!-\!\!\bullet\ g$	4
$f \bullet\!\!-\!\!\bullet\ g$	4

Solución del ejemplo 6.20

Algoritmo de Prim:

En el procedimiento de *Prim* existe en cada una de las iteraciones un conjunto de "aristas seleccionables" que denotaremos con el acrónimo AS. En AS siempre se eligirá el lado siguiente para el árbol T de expansión mínima. Las aristas en el conjunto AS no tienen por qué incorporarse en el orden de V. El conjunto AS se actualiza eliminando el lado de T seleccionado en la iteración anterior, suprimiendo aquellas aristas que forman rutas circuitales en T e incluyendo los lados del grafo incidentes al nuevo nodo de T, también agregado, en la iteración anterior. Al tener actualizado el conjunto AS, el lado a incluir en T, debe satisfacer tener el menor peso. Se aprecia, además, que al seleccionar una arista que se encuentra en el conjunto AS, se implica cumplir con el requisito, establecido en el algoritmo de *Prim*, de elegir un lado incidente a alguno de los vértices de T. A continuación, se describen cada una de las iteraciones involucradas en este ejemplo.

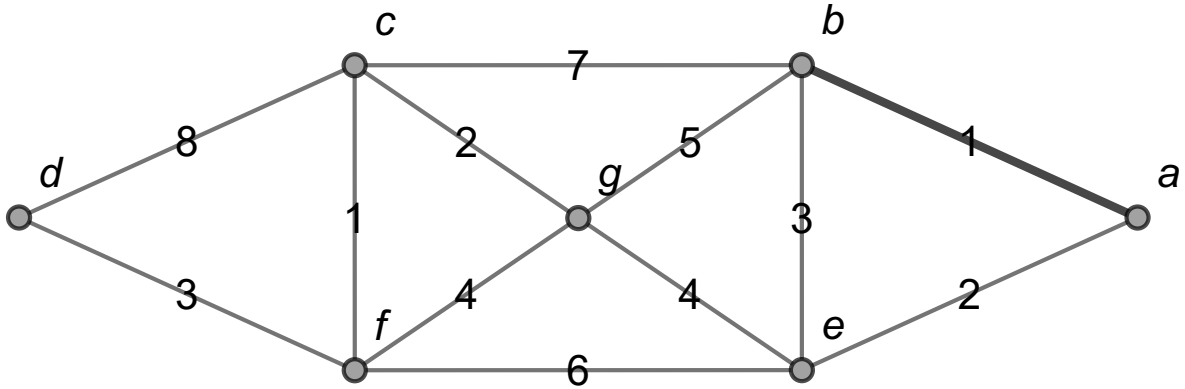
Solución del ejemplo 6.20

- Inicialmente:

$$AS = \left\{ \underset{1}{[a,b]}, \underset{2}{[a,e]} \right\}$$

Los números corresponden a la ponderación de cada arista en el grafo. Se escoge el lado $[a,b]$ de menor peso 1 y se agrega al árbol de expansión T :

$$T = \left\{ \underset{1}{[a,b]} \right\}$$



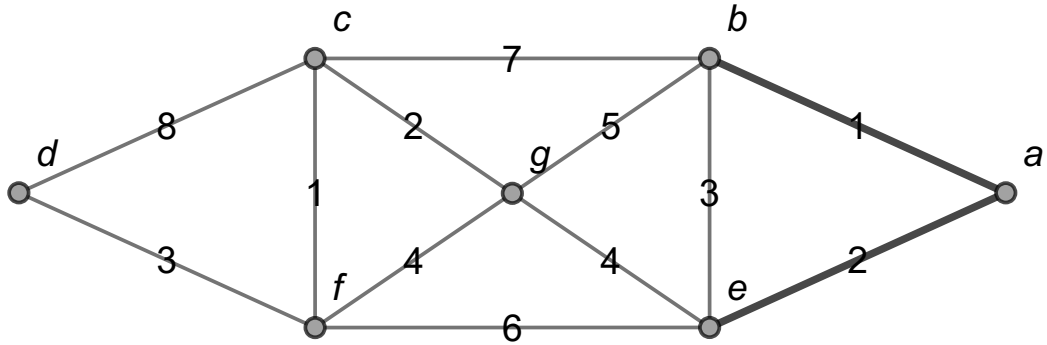
Solución del ejemplo 6.20

- Se actualiza AS, eliminando el lado $[a, b]$ y añadiendo todas las aristas nuevas incidentes a b en el grafo:

$$AS = \left\{ \underset{2}{[a, e]}, \underset{7}{[b, c]}, \underset{3}{[b, e]}, \underset{5}{[b, g]} \right\}$$

Se elije la arista $[a, e]$ pues tiene el menor peso 2 :

$$T = \left\{ \underset{1}{[a, b]}, \underset{2}{[a, e]} \right\}$$



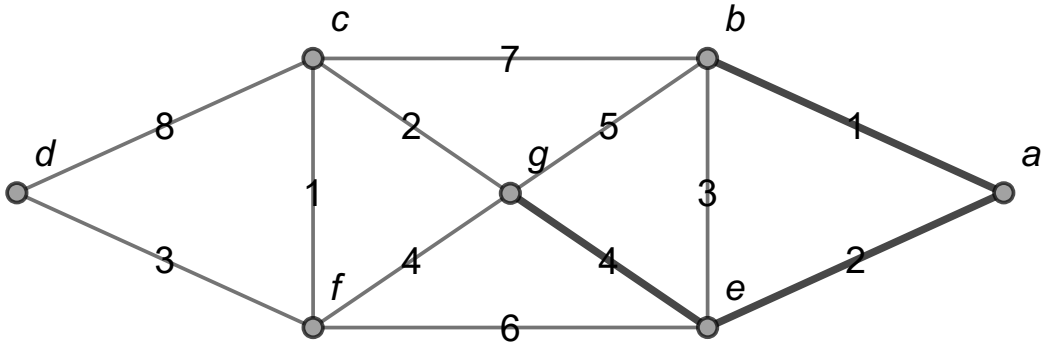
Solución del ejemplo 6.20

- Se elimina de AS el lado $[a, e]$, se suprime de AS la arista $[b, e]$ al formar un circuito en T y se incluyen todas las nuevas aristas en el grafo incidentes a e :

$$AS = \left\{ \underset{7}{[b, c]}, \underset{5}{[b, g]}, \underset{6}{[e, f]}, \underset{4}{[e, g]} \right\}$$

El lado de menor peso es $[e, g]$:

$$T = \left\{ \underset{1}{[a, b]}, \underset{2}{[a, e]}, \underset{4}{[e, g]} \right\}$$



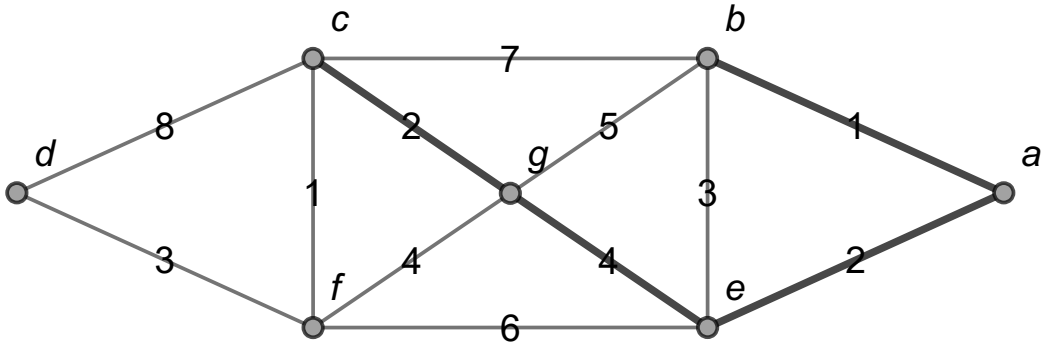
Solución del ejemplo 6.20

- Se suprimen $[e, g]$ y $[b, g]$ de AS. Se elimina $[b, g]$ al generar un circuito en T. Además, se incluyen las nuevas aristas del grafo incidentes a g :

$$AS = \left\{ \underset{7}{[b, c]}, \underset{6}{[e, f]}, \underset{2}{[g, c]}, \underset{4}{[g, f]} \right\}$$

El lado $[g, c]$ de AS tiene el menor peso 2, se añade a T :

$$T = \left\{ \underset{1}{[a, b]}, \underset{2}{[a, e]}, \underset{4}{[e, g]}, \underset{2}{[g, c]} \right\}$$



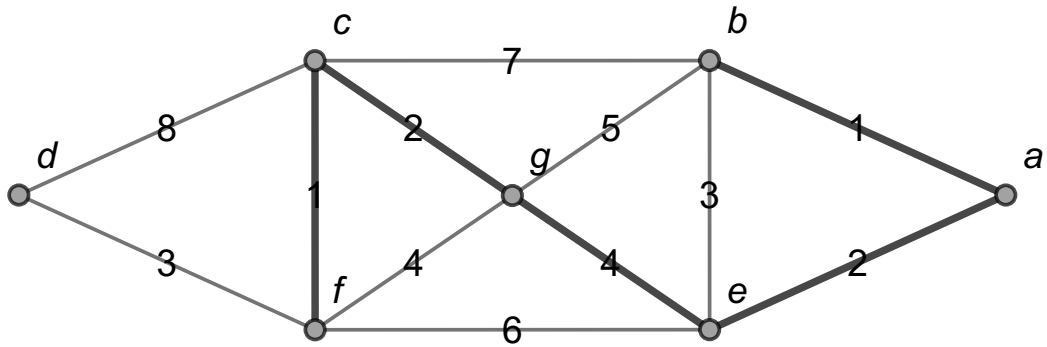
Solución del ejemplo 6.20

- En consecuencia, se elimina $[g, c]$ de AS. También, se omite $[b, c]$ al formar un circuito en T. Luego, se agregan a AS los nuevos lados incidentes al nodo c :

$$AS = \left\{ \underset{6}{[e, f]}, \underset{4}{[g, f]}, \underset{8}{[c, d]}, \underset{1}{[c, f]} \right\}$$

Ahora, se escoge el lado $[c, f]$ pues contiene el menor peso 1 :

$$T = \left\{ \underset{1}{[a, b]}, \underset{2}{[a, e]}, \underset{4}{[e, g]}, \underset{2}{[g, c]}, \underset{1}{[c, f]} \right\}$$



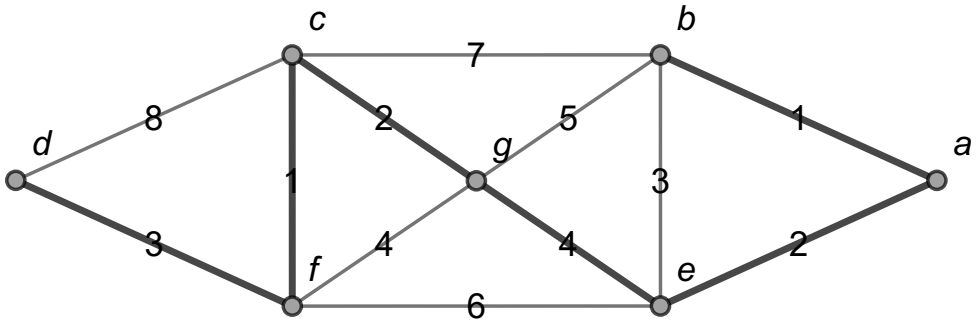
Solución del ejemplo 6.20

- En el conjunto AS se suprimen $[c, f]$, $[e, f]$ y $[g, f]$, estas dos últimas al generar circuitos en T. Ahora, se incluyen los nuevos lados del grafo incidentes a f (solo hay uno):

$$AS = \left\{ \underset{8}{[c, d]}, \underset{3}{[f, d]} \right\}$$

Se selecciona la arista $[f, d]$ al tener la menor ponderación 3 :

$$T = \left\{ \underset{1}{[a, b]}, \underset{2}{[a, e]}, \underset{4}{[e, g]}, \underset{2}{[g, c]}, \underset{1}{[c, f]}, \underset{3}{[f, d]} \right\}$$



Solución del ejemplo 6.20

Ya se ha finalizado pues T posee 6 lados. El peso mínimo del árbol de expansión encontrado, corresponde a la suma de las ponderaciones vinculadas con las aristas del conjunto T, es decir:

$$1 + 2 + 4 + 2 + 1 + 3 = 13$$

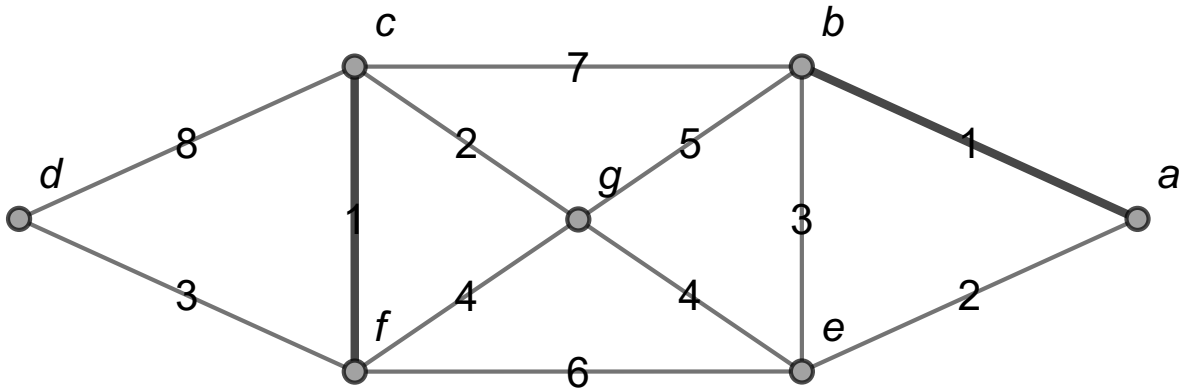
Afirmar que 13 es el peso de cualquier árbol generador minimal, significa que todo árbol recubridor en el grafo, tendrá siempre una ponderación acumulada mayor o igual a 13.

Solución del ejemplo 6.20

Algoritmo de Kruskal:

- Se inicia suponiendo que se tiene en el árbol T todos los vértices del grafo y ningún lado. En el grafo, existen lados de peso 1. Se seleccionan al no formar circuitos en T y se colocan en cualquier orden:

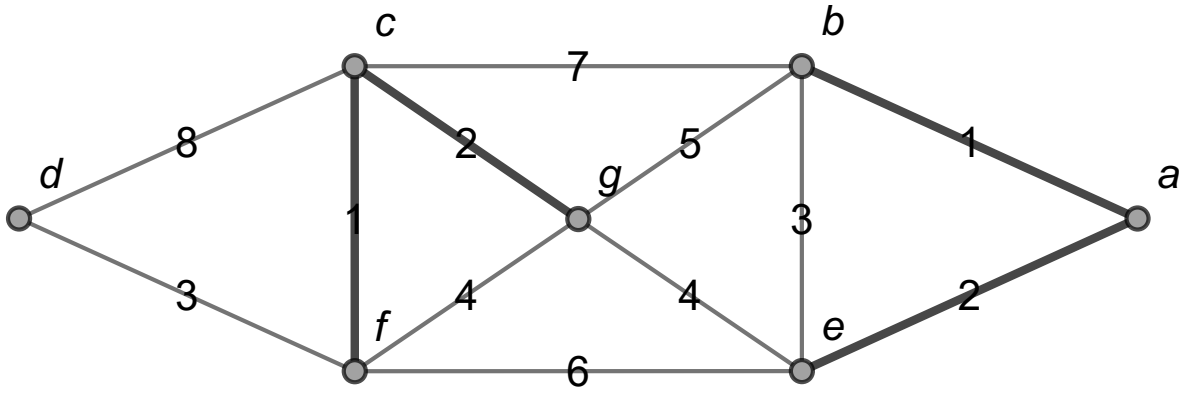
$$T = \left\{ \underset{1}{[c,f]}, \underset{1}{[b,a]} \right\}$$



Solución del ejemplo 6.20

- Ahora, se continua con las aristas de peso 2. Ninguna forma un circuito, por lo que, se agregan al árbol T :

$$T = \left\{ \underset{1}{[c,f]}, \underset{1}{[b,a]}, \underset{2}{[c,g]}, \underset{2}{[e,a]} \right\}$$



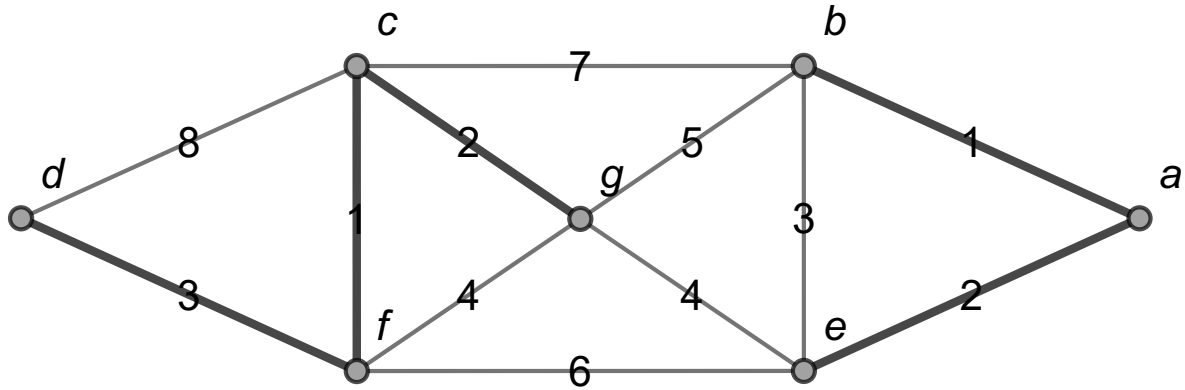
Nota

Como se aprecia, en el algoritmo de *Kruskal* el orden de los vértices no interesa al incluir un lado en el conjunto T. En cambio, en el algoritmo de *Prim*, si se invierte el orden de los nodos al incorporar una arista en T, puede ocasionar un error en las iteraciones subsiguientes.

Solución del ejemplo 6.20

- En T, hay dos lados con ponderación igual a 3 : $[b, e]$ y $[f, d]$. Dado que $[b, e]$ genera una ruta circuital, se elige únicamente a $[f, d]$ y se añade a T :

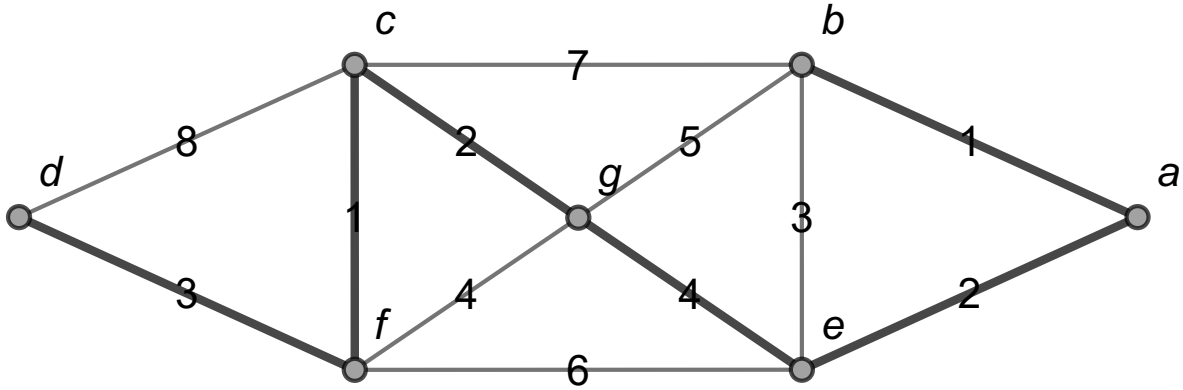
$$T = \left\{ \underset{1}{[c, f]}, \underset{1}{[b, a]}, \underset{2}{[c, g]}, \underset{2}{[e, a]}, \underset{3}{[f, d]} \right\}$$



Solución del ejemplo 6.20

- Finalmente, se incluye en T el lado $[g, e]$ de peso 4. En el grafo existe otra arista de ponderación 4, el lado $[g, f]$, sin embargo, se descarta pues forma un circuito en T. Luego:

$$T = \left\{ \underset{1}{[c, f]}, \underset{1}{[b, a]}, \underset{2}{[c, g]}, \underset{2}{[e, a]}, \underset{3}{[f, d]}, \underset{4}{[g, e]} \right\}$$



Nota

El método de *Kruskal* ha retornado el mismo árbol de expansión mínimo en comparación con el algoritmo de *Prim*, aunque en un orden distinto. Algunas veces, como se mostrará en el siguiente ejemplo, *Kruskal* facilita hallar otros árboles recubridores minimales distintos al obtenido con el procedimiento de *Prim*.

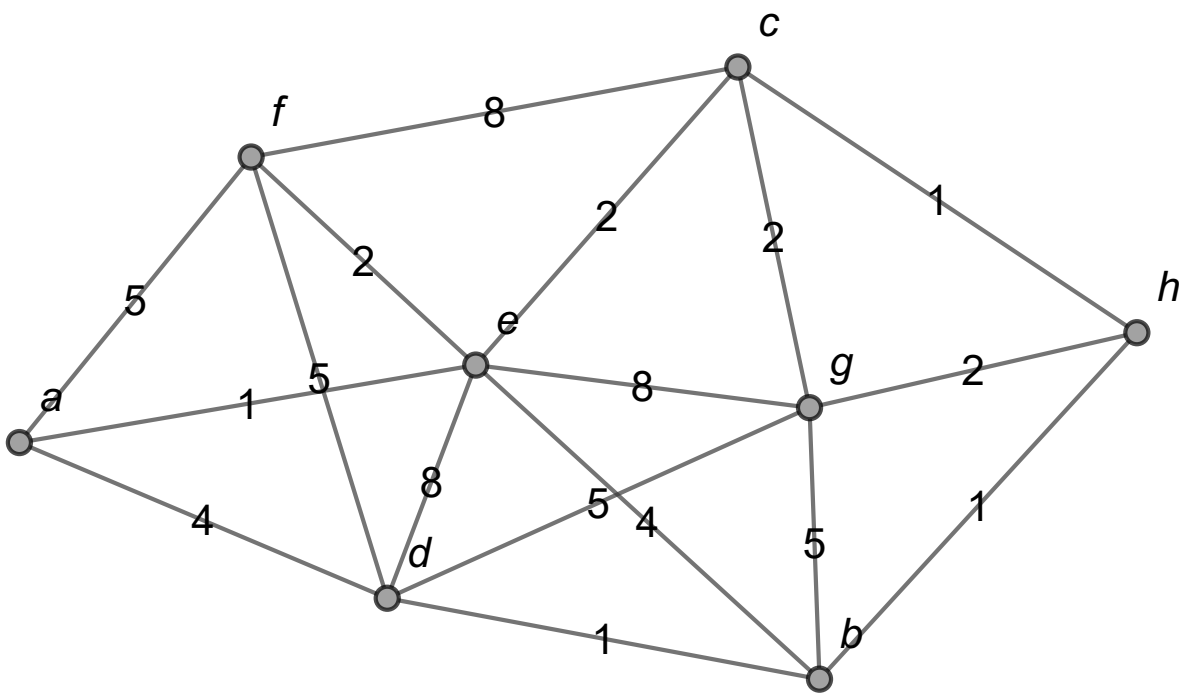
Por otra parte, de manera equivalente a lo señalado en los algoritmos "buscar primero a lo ancho" y "buscar primero a lo largo", cuando se aplican los métodos de *Prim* y de *Kruskal* sobre un grafo conexo, la respuesta del ejercicio se circunscribe netamente en el conjunto T y no en la representación gráfica del árbol generador. Estas figuras se han mostrado en la solución solo con fines didácticos.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-156.zip

Example (6.21)

Halle recurriendo a los métodos de *Prim* y de *Kruskal*, un árbol de expansión mínima en el siguiente grafo, suponiendo el orden del abecedario sobre el conjunto V de vértices.



Solución del ejemplo 6.21

Al visualizar el grafo se deduce la siguiente tabla de ponderaciones:

Arista	Peso
$a \bullet\!\!-\!\!\bullet\ d$	4
$a \bullet\!\!-\!\!\bullet\ e$	1
$a \bullet\!\!-\!\!\bullet\ f$	5
$b \bullet\!\!-\!\!\bullet\ h$	1
$d \bullet\!\!-\!\!\bullet\ b$	1
$d \bullet\!\!-\!\!\bullet\ f$	5
$d \bullet\!\!-\!\!\bullet\ g$	5
$e \bullet\!\!-\!\!\bullet\ b$	4
$e \bullet\!\!-\!\!\bullet\ c$	2

Arista	Peso
$e \bullet\!\!-\!\!\bullet\ d$	8
$e \bullet\!\!-\!\!\bullet\ f$	2
$e \bullet\!\!-\!\!\bullet\ g$	8
$f \bullet\!\!-\!\!\bullet\ c$	8
$g \bullet\!\!-\!\!\bullet\ b$	5
$g \bullet\!\!-\!\!\bullet\ c$	2
$h \bullet\!\!-\!\!\bullet\ c$	1
$h \bullet\!\!-\!\!\bullet\ g$	2

El número n de nodos es $n = 8$, por lo cual, *Prim* y *Kruskal* terminan su ejecución cuando el árbol T a construir, posea una cantidad de aristas igual a $n - 1 = 8 - 1 = 7$.

Solución del ejemplo 6.21

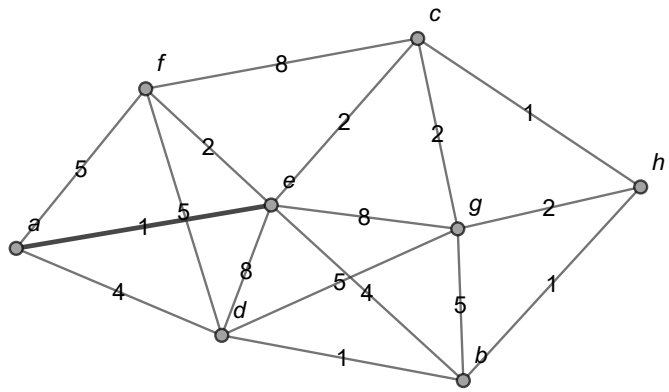
Método de Prim:

- En un inicio:

$$AS = \left\{ \underset{4}{[a,d]}, \underset{1}{[a,e]}, \underset{5}{[a,f]} \right\}$$

Se selecciona el lado $[a,e]$ de menor ponderación y se añade al árbol T :

$$T = \left\{ \underset{1}{[a,e]} \right\}$$



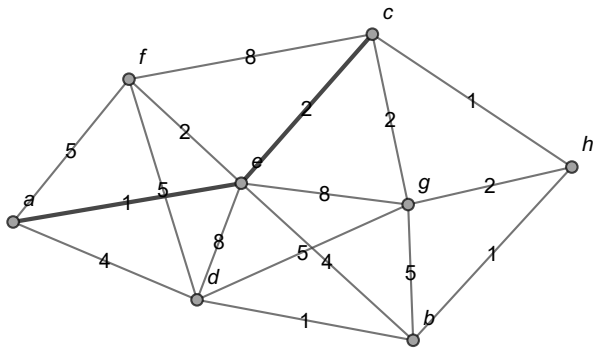
Solución del ejemplo 6.21

- Se elimina $[a, e]$ del conjunto AS y se incluyen allí, las nuevas aristas del grafo incidentes a e :

$$AS = \left\{ \underset{4}{[a,d]}, \underset{5}{[a,f]}, \underset{4}{[e,b]}, \underset{2}{[e,c]}, \underset{8}{[e,d]}, \underset{2}{[e,f]}, \underset{8}{[e,g]} \right\}$$

Existen dos lados de menor peso: $[e, \boxed{c}]$ y $\left[e, \boxed{f}\right]$. De acuerdo al orden del alfabeto, corresponde elegir a $[e, c]$:

$$T = \left\{ \underset{1}{[a,e]}, \underset{2}{[e,c]} \right\}$$



Nota

Como se observa, en el algoritmo de *Prim* el orden de partida para los vértices del grafo, se utiliza con el objetivo de resolver empates en las ponderaciones mínimas.

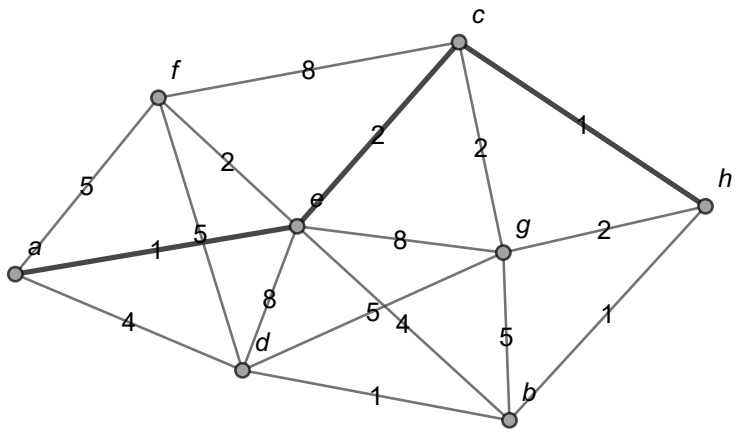
Solución del ejemplo 6.21

- Se suprime de AS el lado $[e, c]$ y se adicionan las nuevas aristas incidentes al nodo c :

$$AS = \left\{ \underset{4}{[a,d]}, \underset{5}{[a,f]}, \underset{4}{[e,b]}, \underset{8}{[e,d]}, \underset{2}{[e,f]}, \underset{8}{[e,g]}, \underset{8}{[c,f]}, \underset{2}{[c,g]}, \underset{1}{[c,h]} \right\}$$

El lado $[c, h]$ de peso 1 se agrega a T :

$$T = \left\{ \underset{1}{[a,e]}, \underset{2}{[e,c]}, \underset{1}{[c,h]} \right\}$$



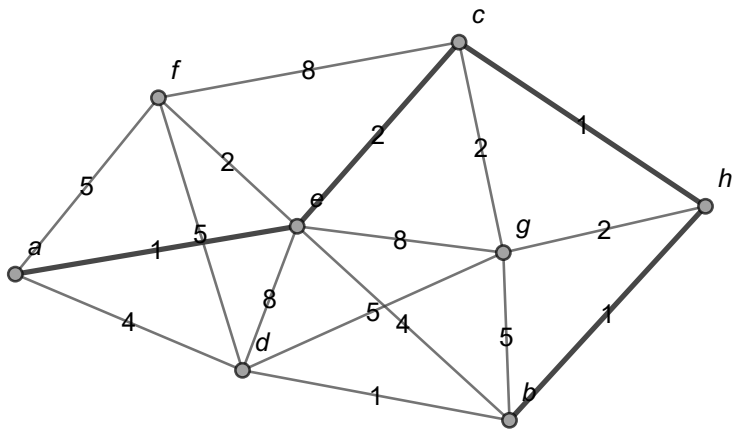
Solución del ejemplo 6.21

- Se elimina $[c, h]$ de AS y se agregan las nuevas aristas del grafo incidentes a h :

$$AS = \left\{ \underset{4}{[a,d]}, \underset{5}{[a,f]}, \underset{4}{[e,b]}, \underset{8}{[e,d]}, \underset{2}{[e,f]}, \underset{8}{[e,g]}, \underset{8}{[c,f]}, \underset{2}{[c,g]}, \underset{1}{[h,b]}, \underset{2}{[h,g]} \right\}$$

Se escoge el lado $[h, b]$:

$$T = \left\{ \underset{1}{[a,e]}, \underset{2}{[e,c]}, \underset{1}{[c,h]}, \underset{1}{[h,b]} \right\}$$



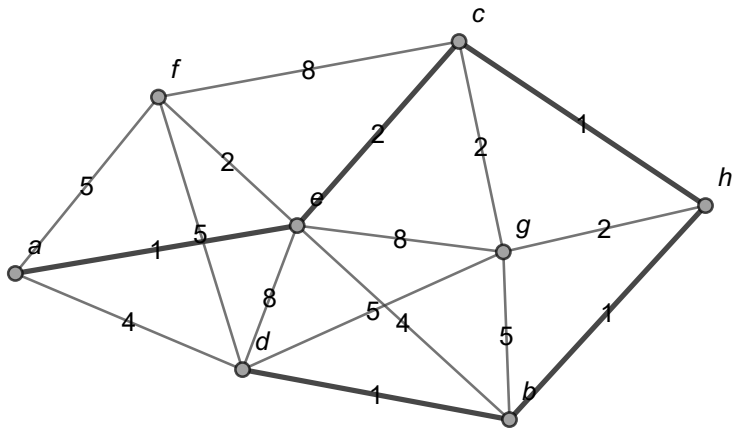
Solución del ejemplo 6.21

- En AS, se omite $[h, b]$ y $[e, b]$, esta última por formar un circuito en T. Además, se añaden los nuevos lados incidentes a b :

$$AS = \left\{ \underset{4}{[a,d]}, \underset{5}{[a,f]}, \underset{8}{[e,d]}, \underset{2}{[e,f]}, \underset{8}{[e,g]}, \underset{8}{[c,f]}, \underset{2}{[c,g]}, \underset{2}{[h,g]}, \underset{1}{[b,d]}, \underset{5}{[b,g]} \right\}$$

Se agrega a T el lado $[b, d]$ de menor ponderación:

$$T = \left\{ \underset{1}{[a,e]}, \underset{2}{[e,c]}, \underset{1}{[c,h]}, \underset{1}{[h,b]}, \underset{1}{[b,d]} \right\}$$



Solución del ejemplo 6.21

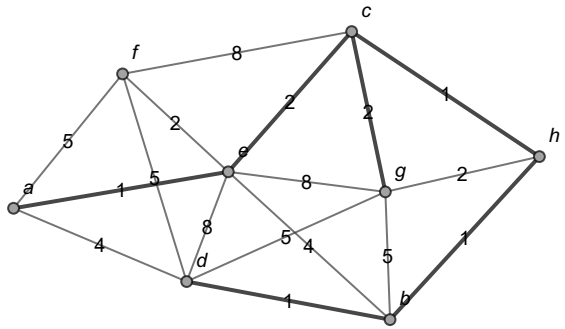
- Ahora, se elimina $[b, d]$, $[a, d]$ y $[e, d]$ de AS, las dos últimas al perfilar circuitos en T y se incorporan en AS las nuevas aristas del grafo con extremo d :

$$AS = \left\{ \underset{5}{[a, f]}, \underset{2}{[e, f]}, \underset{8}{[e, g]}, \underset{8}{[c, f]}, \underset{2}{[c, g]}, \underset{2}{[h, g]}, \underset{5}{[b, g]}, \underset{5}{[d, f]}, \underset{5}{[d, g]} \right\}$$

Existen tres aristas de menor peso igual a 2 : $\left[\boxed{e}, f\right]$, $\left[\boxed{c}, g\right]$ y $\left[\boxed{h}, g\right]$.

Por el orden del abecedario, se elige a $[c, g]$:

$$T = \left\{ \underset{1}{[a, e]}, \underset{2}{[e, c]}, \underset{1}{[c, h]}, \underset{1}{[h, b]}, \underset{1}{[b, d]}, \underset{2}{[c, g]} \right\}$$



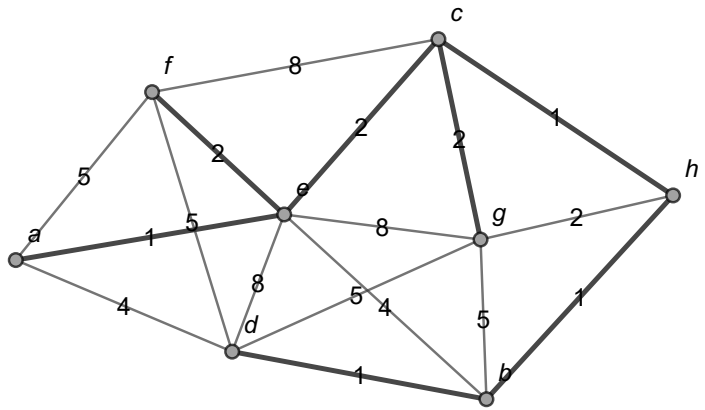
Solución del ejemplo 6.21

- Se suprime de AS el lado $[c, g]$ y se eliminan los lados $[e, g]$, $[h, g]$, $[b, g]$ y $[d, g]$ al generar circuitos en T. No hay nuevas aristas con extremo g :

$$AS = \left\{ \underset{5}{[a, f]}, \underset{2}{[e, f]}, \underset{8}{[c, f]}, \underset{5}{[d, f]} \right\}$$

Corresponde incluir en T el lado $[e, f]$:

$$T = \left\{ \underset{1}{[a, e]}, \underset{2}{[e, c]}, \underset{1}{[c, h]}, \underset{1}{[h, b]}, \underset{1}{[b, d]}, \underset{2}{[c, g]}, \underset{2}{[e, f]} \right\}$$



Solución del ejemplo 6.21

Se ha terminado el proceso pues T contiene 7 lados y el peso mínimo del árbol de expansión hallado es:

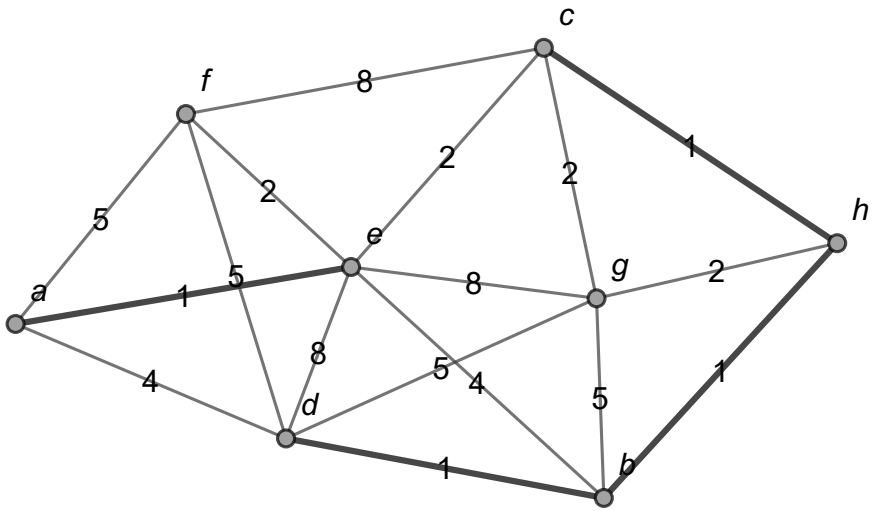
$$1 + 2 + 1 + 1 + 1 + 2 + 2 = 10$$

Solución del ejemplo 6.21

Método de Kruskal:

- Se comienza asumiendo que el árbol recubridor T a construir, incluye todos los nodos del grafo y ninguna arista. En el grafo, existen lados de peso 1. Se agregan todos a T al no generar rutas circuitales y se colocan en cualquier orden:

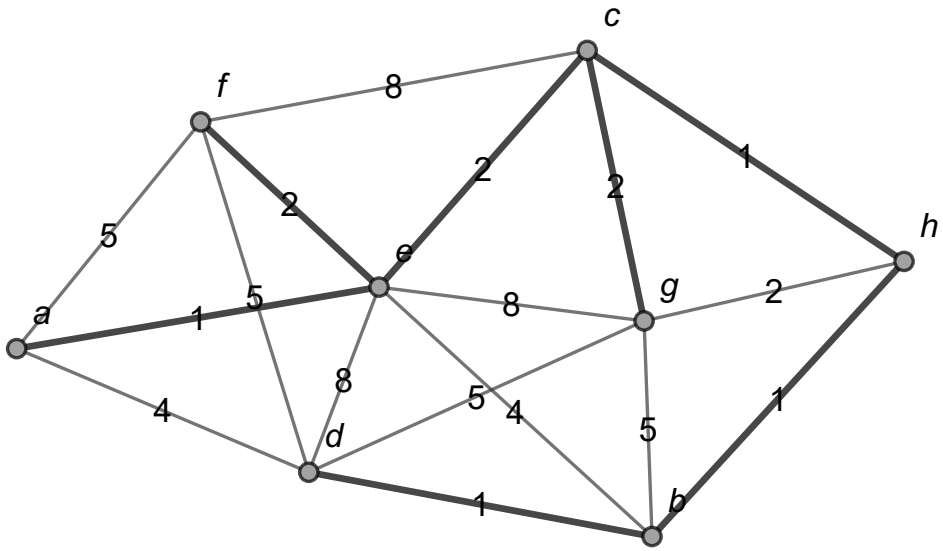
$$T = \left\{ \underset{1}{[d,b]}, \underset{1}{[b,h]}, \underset{1}{[h,c]}, \underset{1}{[e,a]} \right\}$$



Solución del ejemplo 6.21

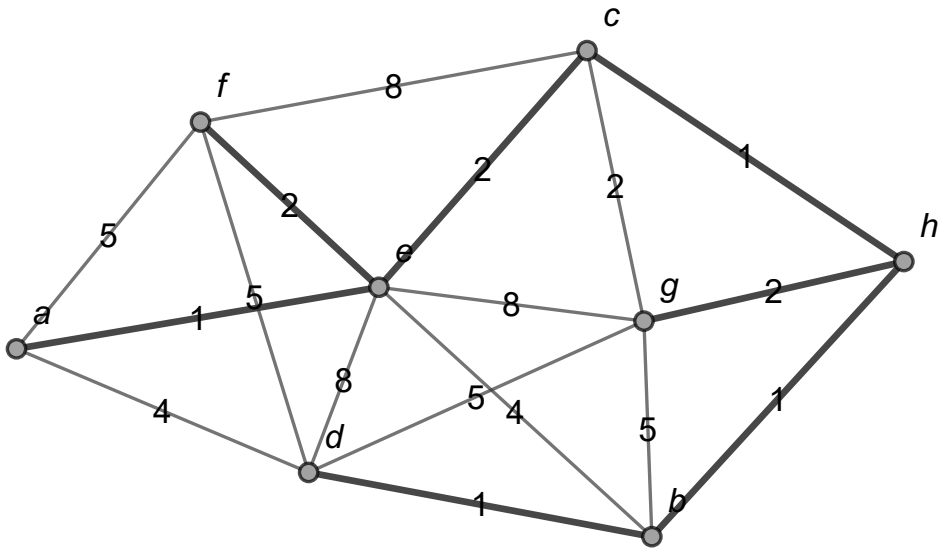
- Ahora, prosigue incorporar a T las aristas de peso 2. En el grafo hay cuatro lados de peso 2 y dado que T contiene 4 aristas en este punto de aplicación del algoritmo, hacen falta solo tres lados más, esto significa que hay varios árboles generadores de peso mínimo retornados por *Kruskal*. Estos árboles son:

$$T = \left\{ \underset{1}{[d,b]}, \underset{1}{[b,h]}, \underset{1}{[h,c]}, \underset{1}{[e,a]}, \underset{2}{[f,e]}, \underset{2}{[c,e]}, \underset{2}{[g,c]} \right\}$$



Solución del ejemplo 6.21

$$T = \left\{ \underset{1}{[d,b]}, \underset{1}{[b,h]}, \underset{1}{[h,c]}, \underset{1}{[e,a]}, \underset{2}{[f,e]}, \underset{2}{[c,e]}, \underset{2}{[g,h]} \right\}$$



Naturalmente, uno de ellos corresponde al árbol de expansión devuelto por *Prim*.

Nota

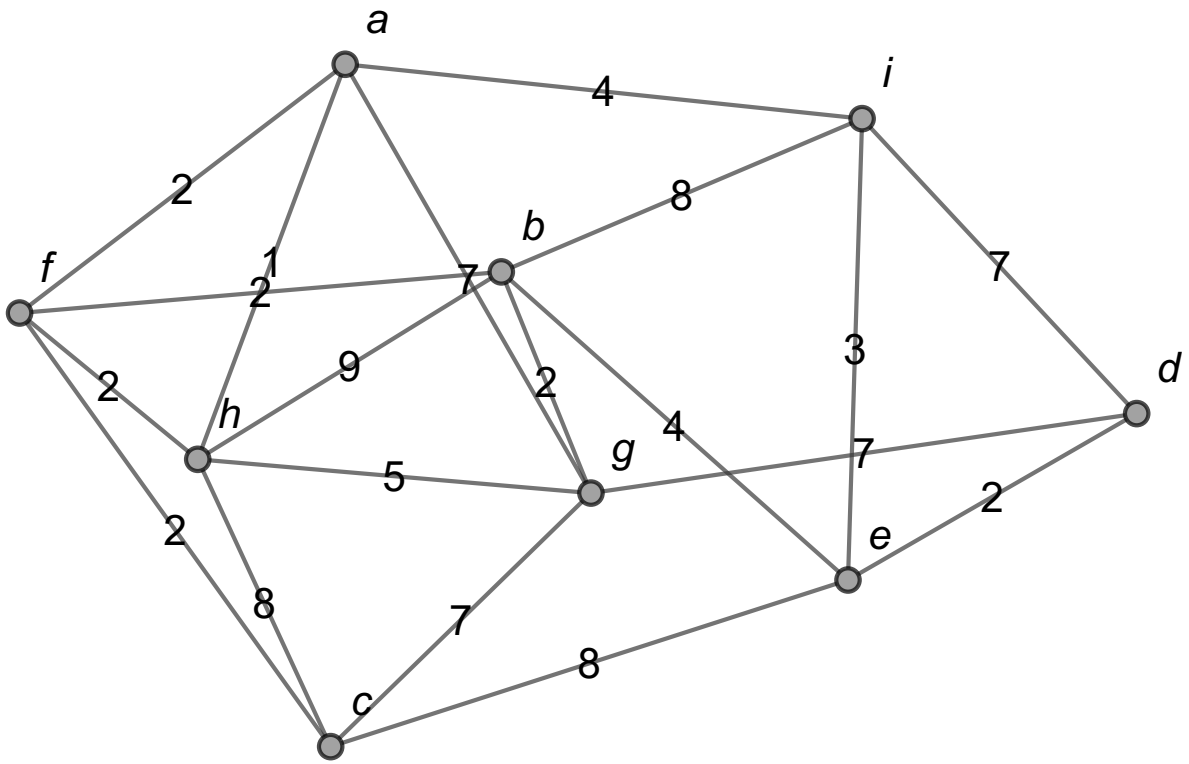
También, cabe destacar que aunque se han determinado dos árboles recubridores distintos mediante *Kruskal*, ambos tienen el mismo peso mínimo igual a 10. Si este valor acumulado no coincide en *Prim* y *Kruskal* es un indicativo de la presencia de algún error.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-157.zip

Example (6.22)

Encuentre un árbol recubridor de peso mínimo en el grafo conexo y ponderado compartido, con el orden $V = \{d, e, f, c, a, i, h, g, b\}$.



Solución del ejemplo 6.22

La siguiente tabla muestra los pesos de cada uno los lados del grafo:

Arista	Peso
$a \bullet\!\!-\!\!\bullet f$	2
$a \bullet\!\!-\!\!\bullet g$	7
$a \bullet\!\!-\!\!\bullet h$	1
$a \bullet\!\!-\!\!\bullet i$	4
$b \bullet\!\!-\!\!\bullet e$	4
$b \bullet\!\!-\!\!\bullet f$	2
$b \bullet\!\!-\!\!\bullet g$	2
$b \bullet\!\!-\!\!\bullet h$	9
$b \bullet\!\!-\!\!\bullet i$	8
$c \bullet\!\!-\!\!\bullet e$	8

Arista	Peso
$c \bullet\!\!-\!\!\bullet f$	2
$c \bullet\!\!-\!\!\bullet g$	7
$c \bullet\!\!-\!\!\bullet h$	8
$d \bullet\!\!-\!\!\bullet e$	2
$d \bullet\!\!-\!\!\bullet g$	7
$d \bullet\!\!-\!\!\bullet i$	7
$e \bullet\!\!-\!\!\bullet i$	3
$h \bullet\!\!-\!\!\bullet f$	2
$h \bullet\!\!-\!\!\bullet g$	5

Los algoritmos de *Prim* y de *Kruskal* finalizan cuando el árbol generador T a construir, contenga $n - 1 = 9 - 1 = 8$ aristas.

Solución del ejemplo 6.22

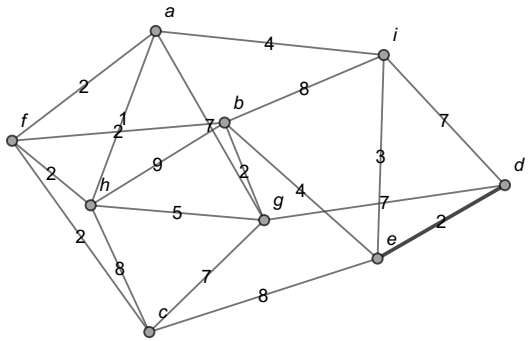
Algoritmo de Prim:

- Se comienza así:

$$AS = \left\{ \underset{2}{[d,e]}, \underset{7}{[d,i]}, \underset{7}{[d,g]} \right\}$$

Aunque en AS se está respetando el orden $V = \{d, e, f, c, a, i, h, g, b\}$, se reitera al estudiante que esto no es necesario dado que los lados en AS pueden tomar cualquier posición. Se elige la arista $[d,e]$ de menor peso y se agrega al árbol T :

$$T = \left\{ \underset{2}{[d,e]} \right\}$$



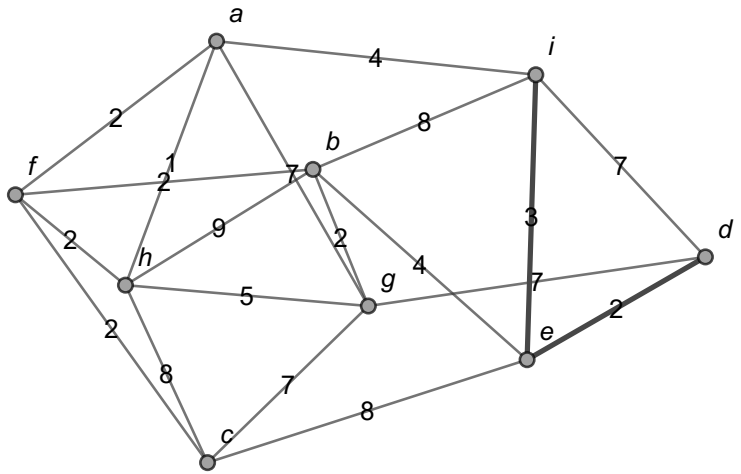
Solución del ejemplo 6.22

- En AS, se elimina $[d, e]$ y se adicionan los nuevos lados del grafo incidentes a e :

$$AS = \left\{ \underset{7}{[d,i]}, \underset{7}{[d,g]}, \underset{8}{[e,c]}, \underset{3}{[e,i]}, \underset{4}{[e,b]} \right\}$$

La arista de menor peso es $[e, i]$:

$$T = \left\{ \underset{2}{[d,e]}, \underset{3}{[e,i]} \right\}$$



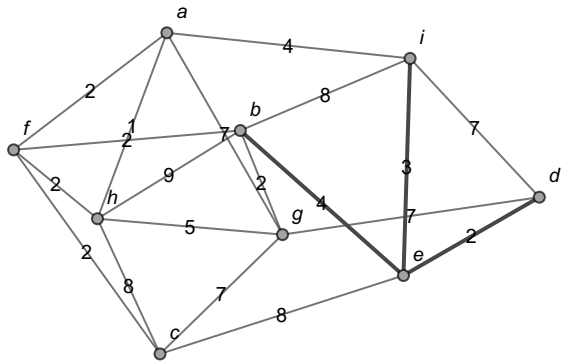
Solución del ejemplo 6.22

- Se suprime $[e, i]$ y $[d, i]$ de AS, esta última al perfilar un circuito en T y se incluyen los nuevos lados incidentes a i :

$$AS = \left\{ \underset{7}{[d, g]}, \underset{8}{[e, c]}, \underset{4}{[e, b]}, \underset{4}{[i, a]}, \underset{8}{[i, b]} \right\}$$

Hay un empate en la ponderación mínima 4 y ocurre en las aristas: $\left[\boxed{e}, b\right]$ y $\left[\boxed{i}, a\right]$. Al observar el orden $V = \left\{d, \boxed{e}, f, c, a, \boxed{i}, h, g, b\right\}$, se escoge el lado $[e, b]$ al aparecer la e antes que la i en V. Luego:

$$T = \left\{ \underset{2}{[d, e]}, \underset{3}{[e, i]}, \underset{4}{[e, b]} \right\}$$



Solución del ejemplo 6.22

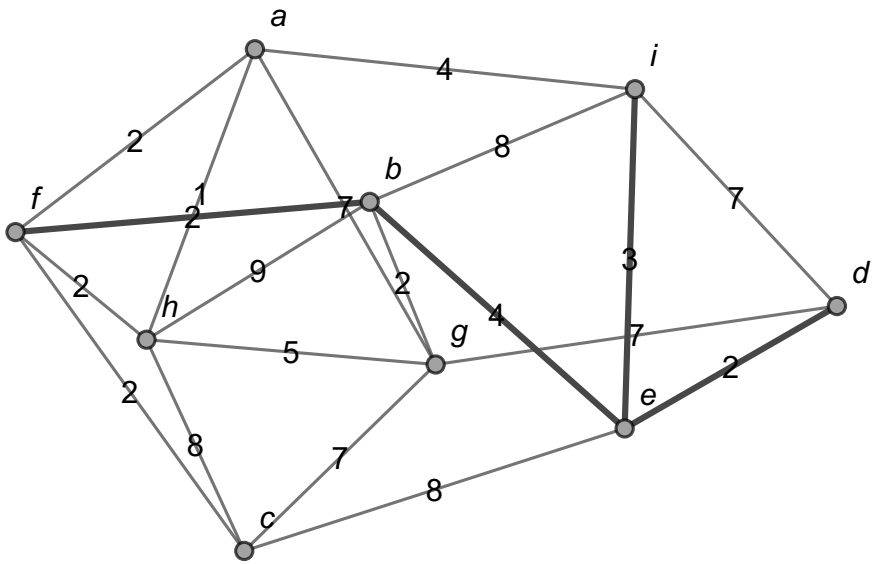
- Ahora, se omiten $[e, b]$ y $[i, b]$ de AS, $[i, b]$ por formar un circuito en T y se añaden a AS, todas las nuevas aristas del grafo incidentes a b :

$$AS = \left\{ \underset{7}{[d, g]}, \underset{8}{[e, c]}, \underset{4}{[i, a]}, \underset{2}{[b, f]}, \underset{9}{[b, h]}, \underset{2}{[b, g]} \right\}$$

Solución del ejemplo 6.22

Los lados $\left[b, \boxed{f}\right]$ y $\left[b, \boxed{g}\right]$ tienen la misma poderación mínima 2 y $V = \left\{d, e, \boxed{f}, c, a, i, h, \boxed{g}, b\right\}$. Se escoge, por lo tanto, la arista $[b, f]$ con la intención de ser agregada a T :

$$T = \left\{ \underset{2}{[d, e]}, \underset{3}{[e, i]}, \underset{4}{[e, b]}, \underset{2}{[b, f]} \right\}$$



Solución del ejemplo 6.22

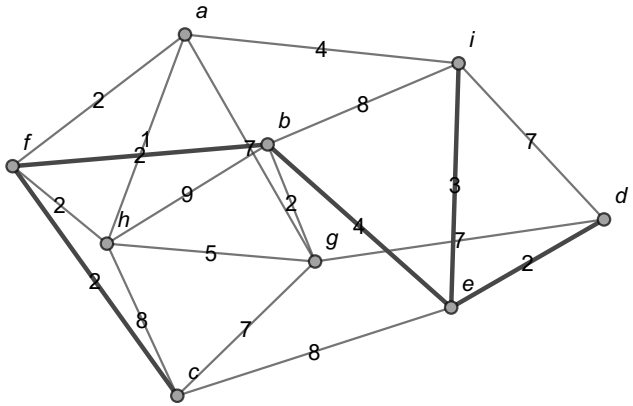
- Se elimina el lado $[b, f]$ de AS y se añaden las nuevas aristas incidentes a f :

$$AS = \left\{ \underset{7}{[d, g]}, \underset{8}{[e, c]}, \underset{4}{[i, a]}, \underset{9}{[b, h]}, \underset{2}{[b, g]}, \underset{2}{[f, c]}, \underset{2}{[f, a]}, \underset{2}{[f, h]} \right\}$$

Solución del ejemplo 6.22

Hay cuatro lados con peso mínimo igual a 2 : $\left[\boxed{b}, g\right]$, $\left[\boxed{f}, c\right]$, $[f, a]$ y $[f, h]$. Como la f está antes que la b en $V = \left\{d, e, \boxed{f}, c, a, i, h, g, \boxed{b}\right\}$, se descarta $[b, g]$. La selección debe realizarse sobre: $[f, \boxed{c}]$, $[f, \boxed{a}]$ y $\left[f, \boxed{h}\right]$. Dado que $V = \left\{d, e, f, \boxed{c}, \boxed{a}, i, \boxed{h}, g, b\right\}$, se elige $[f, c]$:

$$T = \left\{ \underset{2}{[d, e]}, \underset{3}{[e, i]}, \underset{4}{[e, b]}, \underset{2}{[b, f]}, \underset{2}{[f, c]} \right\}$$



Solución del ejemplo 6.22

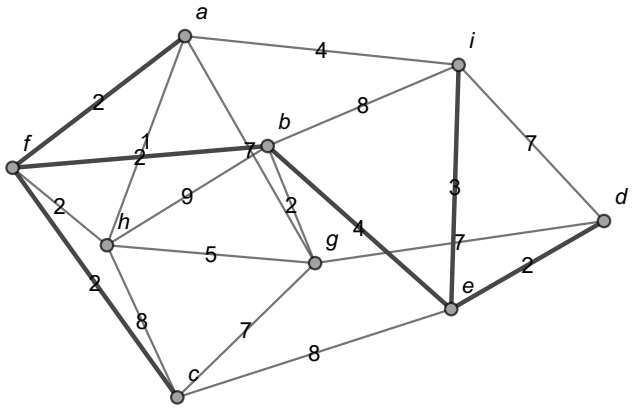
- Ahora, se suprime $[f, c]$ y $[e, c]$ de AS, se omite $[e, c]$ al perfilar un circuito en T y se incorporan a AS, los nuevos lados del grafo incidentes a c :

$$AS = \left\{ \underset{7}{[d, g]}, \underset{4}{[i, a]}, \underset{9}{[b, h]}, \underset{2}{[b, g]}, \underset{2}{[f, a]}, \underset{2}{[f, h]}, \underset{8}{[c, h]}, \underset{7}{[c, g]} \right\}$$

Solución del ejemplo 6.22

Los lados de peso mínimo 2 son: $\left[\boxed{b}, g\right]$, $\left[\boxed{f}, a\right]$ y $[f, h]$, de ellos se descarta $[b, g]$ por el orden $V = \left\{d, e, \boxed{f}, c, a, i, h, g, \boxed{b}\right\}$. Con relación a $[f, \boxed{a}]$ y $\left[f, \boxed{h}\right]$, se observa $V = \left\{d, e, f, c, \boxed{a}, i, \boxed{h}, g, b\right\}$, como consecuencia, se agrega a T la arista $[f, a]$:

$$T = \left\{ \underset{2}{[d, e]}, \underset{3}{[e, i]}, \underset{4}{[e, b]}, \underset{2}{[b, f]}, \underset{2}{[f, c]}, \underset{2}{[f, a]} \right\}$$



Solución del ejemplo 6.22

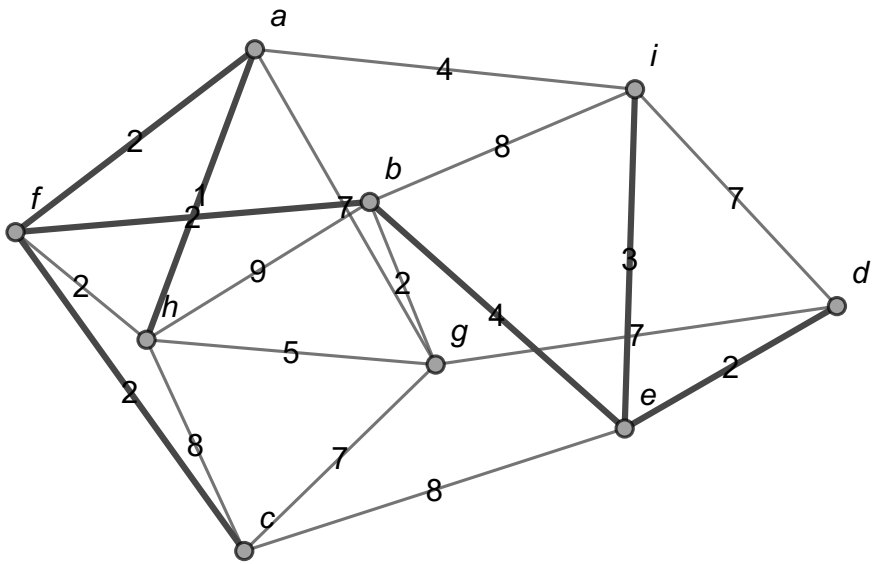
- Se eliminan $[f, a]$ y $[i, a]$ de AS, esta última por formar una ruta circuital en T y se adicionan las nuevas aristas del grafo incidentes a a :

$$AS = \left\{ \underset{7}{[d, g]}, \underset{9}{[b, h]}, \underset{2}{[b, g]}, \underset{2}{[f, h]}, \underset{8}{[c, h]}, \underset{7}{[c, g]}, \underset{1}{[a, h]}, \underset{7}{[a, g]} \right\}$$

Solución del ejemplo 6.22

Se escoge $[a, h]$ de peso 1 :

$$T = \left\{ \underset{2}{[d, e]}, \underset{3}{[e, i]}, \underset{4}{[e, b]}, \underset{2}{[b, f]}, \underset{2}{[f, c]}, \underset{2}{[f, a]}, \underset{1}{[a, h]} \right\}$$



Solución del ejemplo 6.22

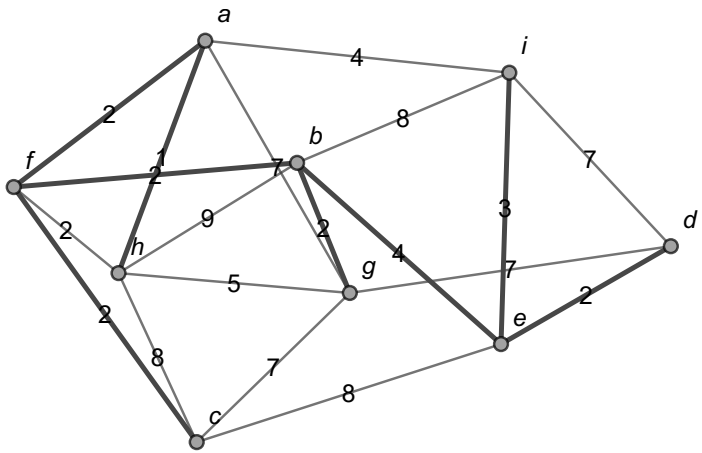
- Se suprimen $[a, h]$, $[b, h]$, $[f, h]$ y $[c, h]$ de AS, las tres últimas por generar circuitos en T y se agregan las nuevas aristas incidentes a h (solo hay una):

$$AS = \left\{ \underset{7}{[d, g]}, \underset{2}{[b, g]}, \underset{7}{[c, g]}, \underset{7}{[a, g]}, \underset{5}{[h, g]} \right\}$$

Solución del ejemplo 6.22

Se selecciona $[b, g]$ con peso mínimo 2 :

$$T = \left\{ \underset{2}{[d, e]}, \underset{3}{[e, i]}, \underset{4}{[e, b]}, \underset{2}{[b, f]}, \underset{2}{[f, c]}, \underset{2}{[f, a]}, \underset{1}{[a, h]}, \underset{2}{[b, g]} \right\}$$



Se ha finalizado pues T posee 8 aristas y el peso mínimo del árbol encontrado corresponde a:

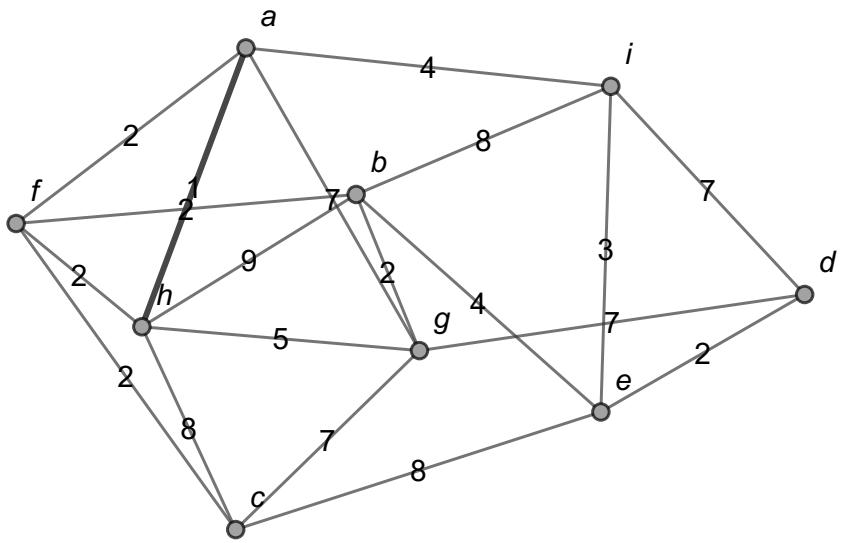
$$2 + 3 + 4 + 2 + 2 + 2 + 1 + 2 = 18$$

Solución del ejemplo 6.22

Algoritmo de Kruskal:

- Se inicia suponiendo que el árbol de expansión T a construir, integra todos los vértices y ninguna arista. En el grafo, hay un lado de peso 1, éste se añade a T :

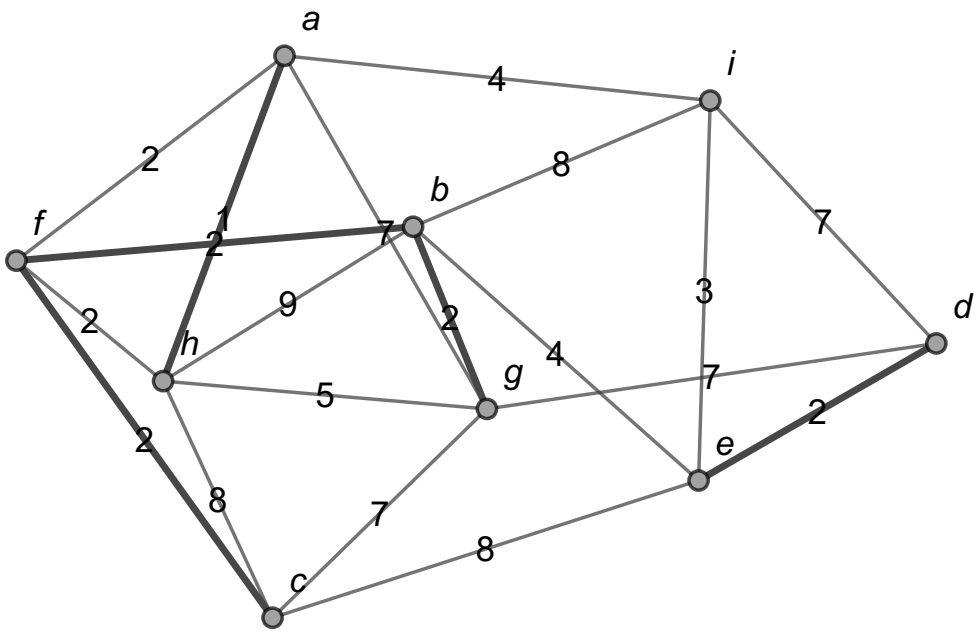
$$T = \left\{ \underset{1}{[h, a]} \right\}$$



Solución del ejemplo 6.22

- Ahora, se incluyen en T las siguientes aristas de peso 2, al no formar circuitos:

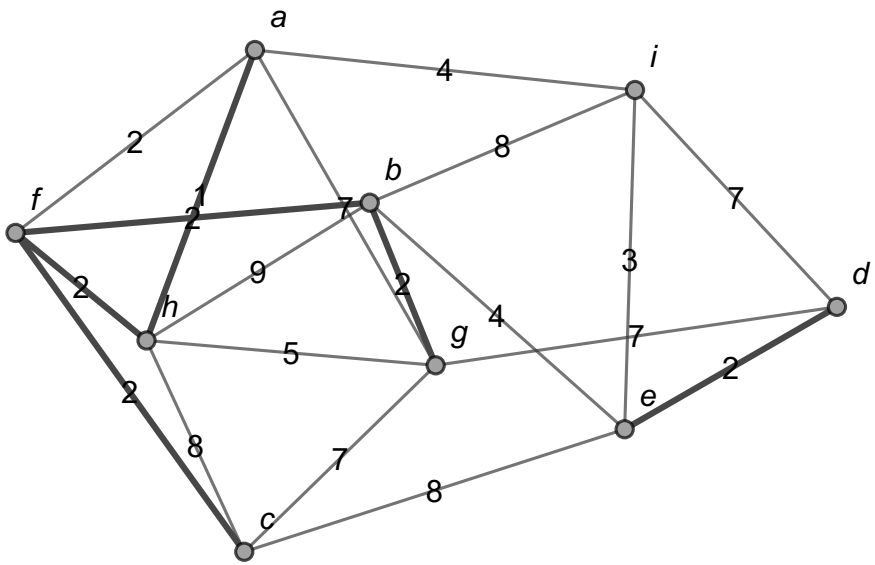
$$T = \left\{ \underset{1}{[h,a]}, \underset{2}{[e,d]}, \underset{2}{[f,c]}, \underset{2}{[b,g]}, \underset{2}{[f,b]} \right\}$$



Solución del ejemplo 6.22

En este punto, hay dos opciones a escoger: $[f, h]$, o bien, $[a, f]$. Ambos lados no se pueden agregar simultáneamente a T pues al seleccionar uno de ellos, el otro genera una ruta circuital en T. Continuan, por lo tanto, dos posibles árboles de expansión mínima:

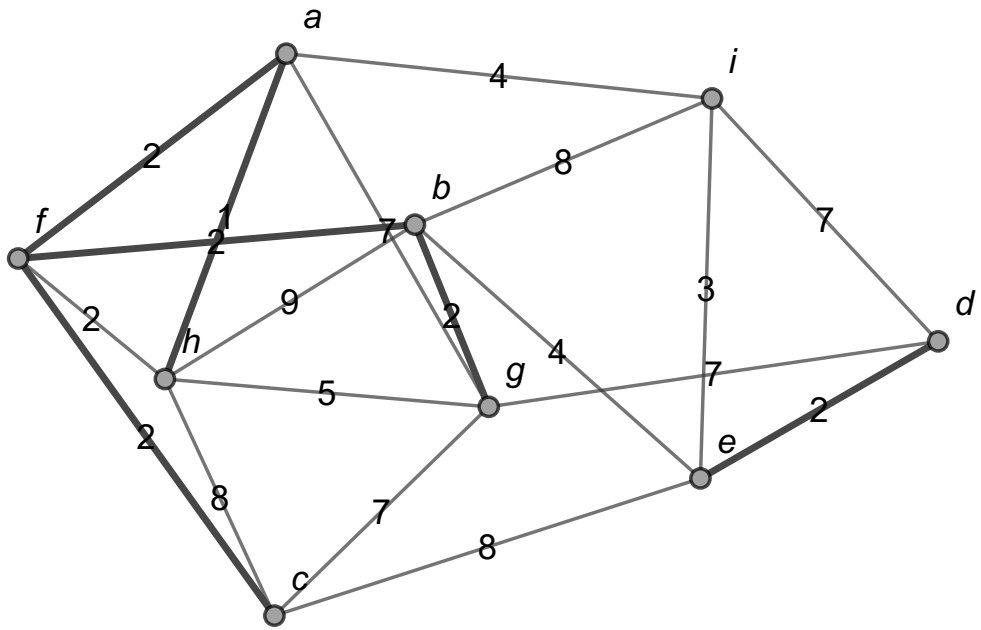
$$T = \left\{ \underset{1}{[h, a]}, \underset{2}{[e, d]}, \underset{2}{[f, c]}, \underset{2}{[b, g]}, \underset{2}{[f, b]}, \underset{2}{[f, h]} \right\}$$



Solución del ejemplo 6.22

O bien:

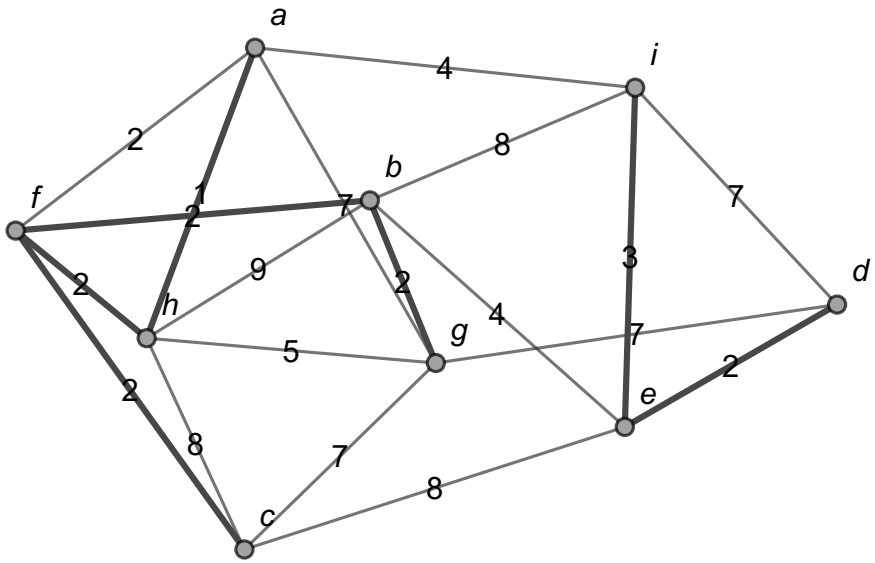
$$T = \left\{ \underset{1}{[h,a]}, \underset{2}{[e,d]}, \underset{2}{[f,c]}, \underset{2}{[b,g]}, \underset{2}{[f,b]}, \underset{2}{[a,f]} \right\}$$



Solución del ejemplo 6.22

- Se prosigue agregando la única arista de peso 3 :

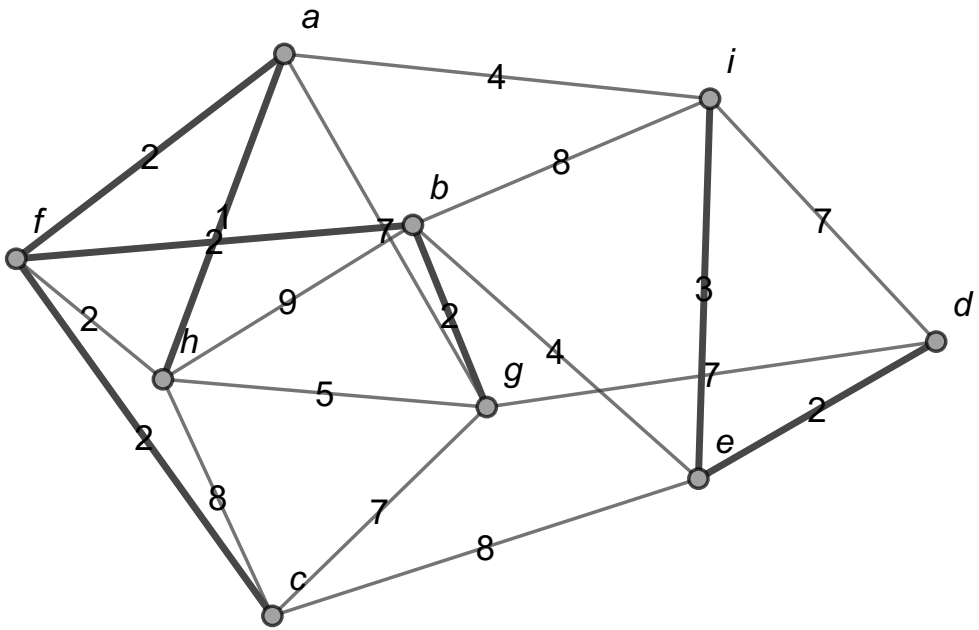
$$T = \left\{ \underset{1}{[h,a]}, \underset{2}{[e,d]}, \underset{2}{[f,c]}, \underset{2}{[b,g]}, \underset{2}{[f,b]}, \underset{2}{[f,h]}, \underset{3}{[e,i]} \right\}$$



Solución del ejemplo 6.22

O bien:

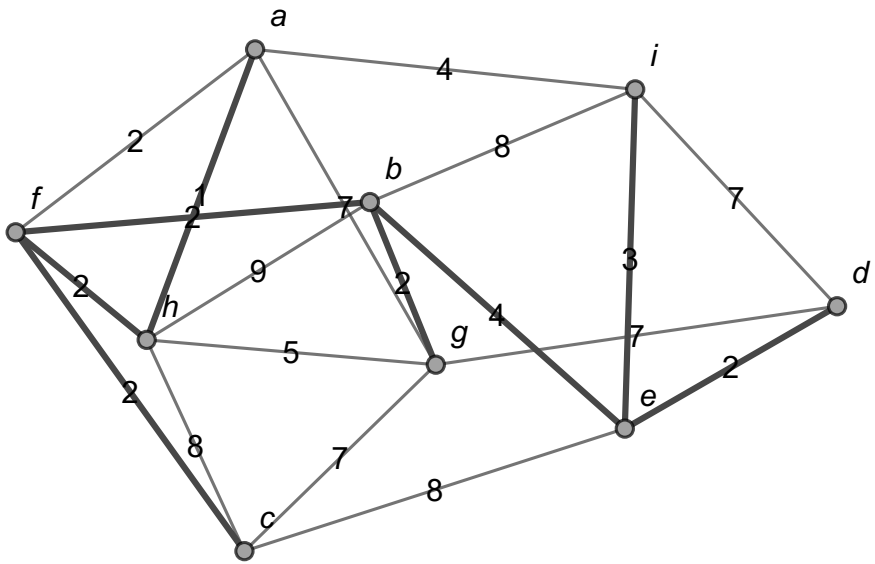
$$T = \left\{ \underset{1}{[h,a]}, \underset{2}{[e,d]}, \underset{2}{[f,c]}, \underset{2}{[b,g]}, \underset{2}{[f,b]}, \underset{2}{[a,f]}, \underset{3}{[e,i]} \right\}$$



Solución del ejemplo 6.22

- Al poseer cada conjunto T anterior, una cardinalidad igual a 7, falta añadir una arista más para terminar el proceso. Finalmente, se incluye el único lado de peso 4 al no perfilar un circuito en ambos árboles:

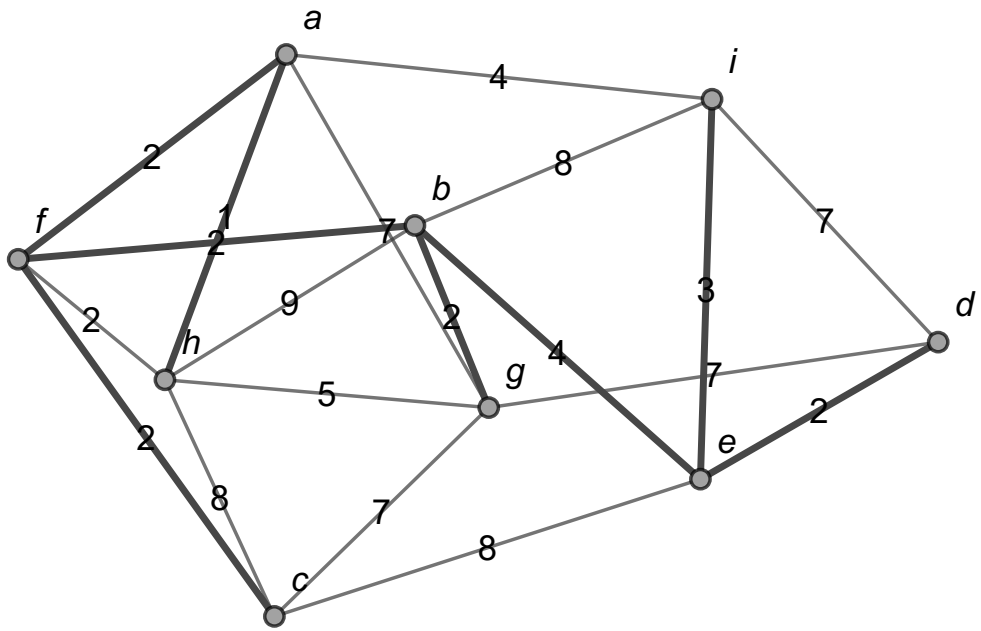
$$T = \left\{ \underset{1}{[h,a]}, \underset{2}{[e,d]}, \underset{2}{[f,c]}, \underset{2}{[b,g]}, \underset{2}{[f,b]}, \underset{2}{[f,h]}, \underset{3}{[e,i]}, \underset{4}{[b,e]} \right\}$$



Solución del ejemplo 6.22

O bien,

$$T = \left\{ \underset{1}{[h,a]}, \underset{2}{[e,d]}, \underset{2}{[f,c]}, \underset{2}{[b,g]}, \underset{2}{[f,b]}, \underset{2}{[a,f]}, \underset{3}{[e,i]}, \underset{4}{[b,e]} \right\}$$



Solución del ejemplo 6.22

Los dos árboles generadores minimales posibles, hallados mediante el método de *Kruskal* tienen el mismo peso mínimo igual a 18. El segundo árbol de expansión de peso mínimo ya expuesto, coincide con el obtenido usando el algoritmo de *Prim*, aunque con un orden distinto.
La librería **VilCretas** contiene dos importantes instrucciones llamadas: `Prim` y `Kruskal`. Ellas ejecutan los algoritmos a los que su nombre hace mención, recibiendo como argumento un grafo conexo y ponderado. Las dos presentan el atributo `all -> True` que facilita un retorno detallado de la aplicación de estos métodos.

Solución del ejemplo 6.22

Particularmente, `Prim` también posee la opción `orden` que especifica el orden a emplear en los nodos del grafo. En este ejercicio:

In[ ] :=

```
grafo = Grafo[{{h, f}, {a, f}, {a, g}, {a, h}, {a, i},
{b, e}, {b, f}, {b, g}, {b, h}, {b, i}, {c, e}, {c, f},
{c, g}, {c, h}, {e, i}, {d, e}, {d, g}, {d, i}, {h, g}},
pesos -> {2, 2, 7, 1, 4, 4, 2, 2, 9, 8, 8, 2, 7, 8, 3, 2,
7, 7, 5}];
Prim[grafo, orden -> {d, e, f, c, a, i, h, g, b}]
Kruskal[grafo]
```

Solución del ejemplo 6.22

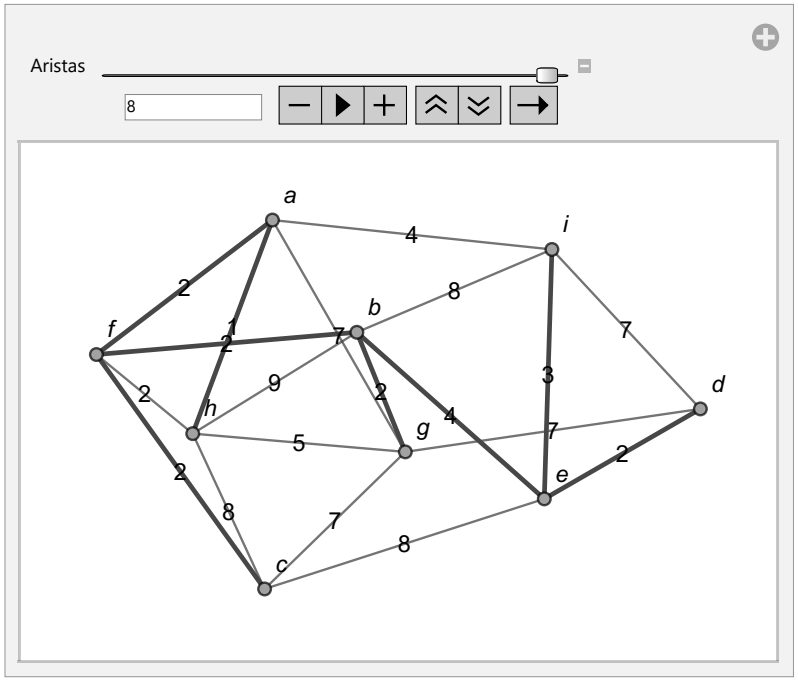
Se obtiene la siguiente salida:

Out[ ] =

El orden de los nodos es: {d, e, f, c, a, i, h, g, b}

T={{d,e}, {e,i}, {e,b}, {b,f}, {f,c}, {f,a}, {a,h}, {b,g}}

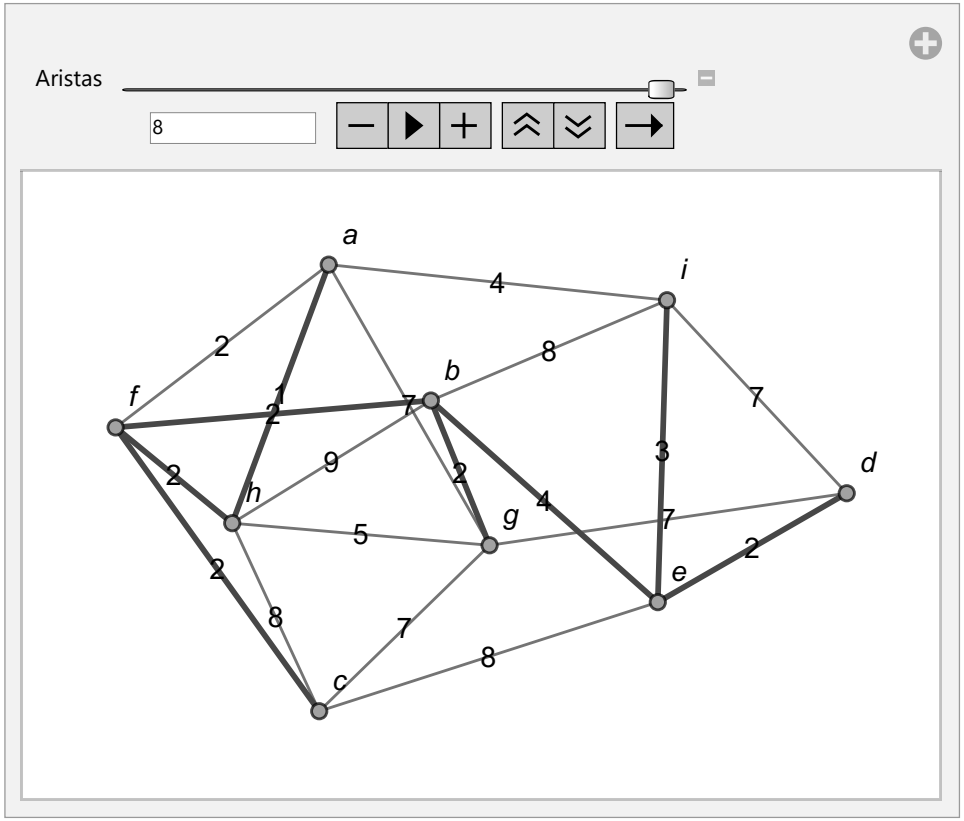
Peso: 18



Solución del ejemplo 6.22

T={a●–●h, b●–●f, b●–●g, d●–●e, h●–●f, c●–●f, e●–●i, b●–●e}

Peso: 18



Solución del ejemplo 6.22

La instrucción `Kruskal` proporciona una salida pseudoaleatoria si existen varios árboles generadores minimales a través de ese procedimiento.

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-158.zip

Utilización del comando `Prim`

Explicación en video

https://youtu.be/DeDKSm26m-w

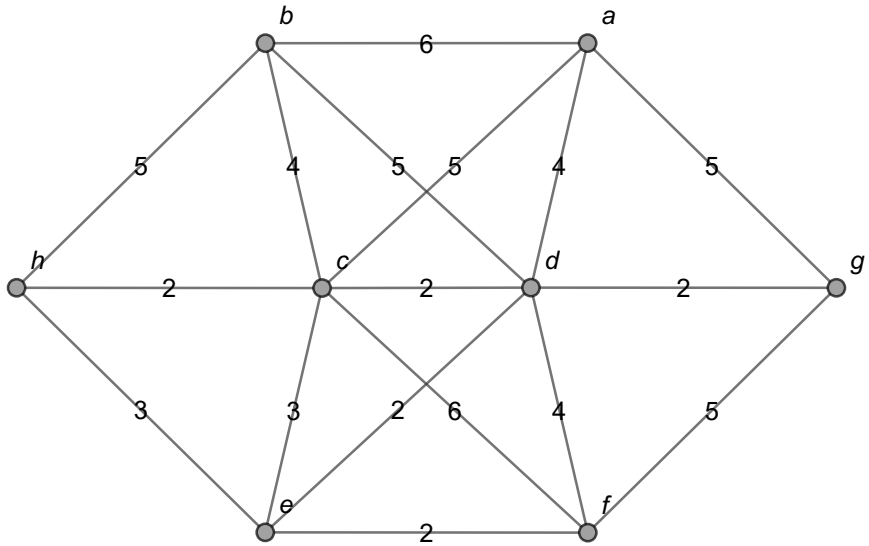
Uso de la sentencia `Kruskal`

Explicación en video

https://youtu.be/TriLaZLAR04

Example (6.23)

Determine con ayuda de *Wolfram Mathematica*, un árbol de expansión de peso máximo con $V = \{b, a, g, e, c, f, h, d\}$, sobre el grafo:



Recurra a los algoritmos de *Prim* y de *Kruskal*.

Solución del ejemplo 6.23

El enunciado del ejemplo solicita encontrar un árbol generador de longitud máxima y tanto *Prim* como *Kruskal* devuelven un árbol de expansión de peso mínimo. Se deben sustituir los pesos del grafo por las mismas poderaciones, solo que, con signo negativo. Al usar los algoritmos de *Prim y de Kruskal* sobre este nuevo grafo, se retornará en cada caso, un árbol recubridor de longitud máxima, en valor absoluto, en lugar de un árbol generador de peso minimal. Luego, en *Wolfram Mathematica*:

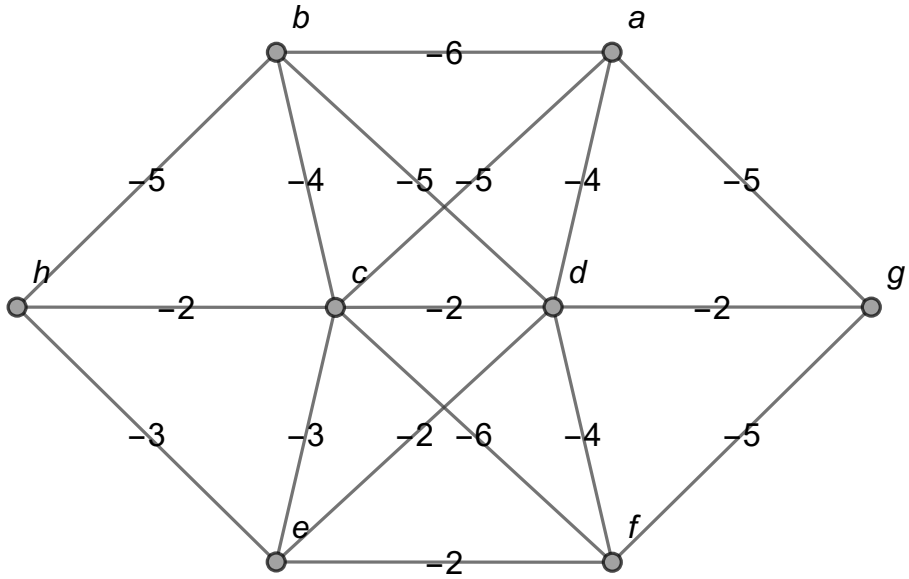
In[ ] :=

```
grafo = Grafo[{{c, e}, {d, e}, {d, a}, {c, d}, {e, h},
{c, h}, {c, f}, {a, b}, {a, c}, {b, c}, {b, d}, {b, h},
{d, f}, {e, f}, {f, g}, {a, g}, {g, d}}, pesos -> -1 {3, 2,
4, 2, 3, 2, 6, 6, 5, 4, 5, 5, 4, 2, 5, 5, 2}, mostrarpesos
-> True]
Prim[grafo, orden -> {b, a, g, e, c, f, h, d}]
Kruskal[grafo]
```

Solución del ejemplo 6.23

Se obtiene la siguiente salida:

Out[ ] =

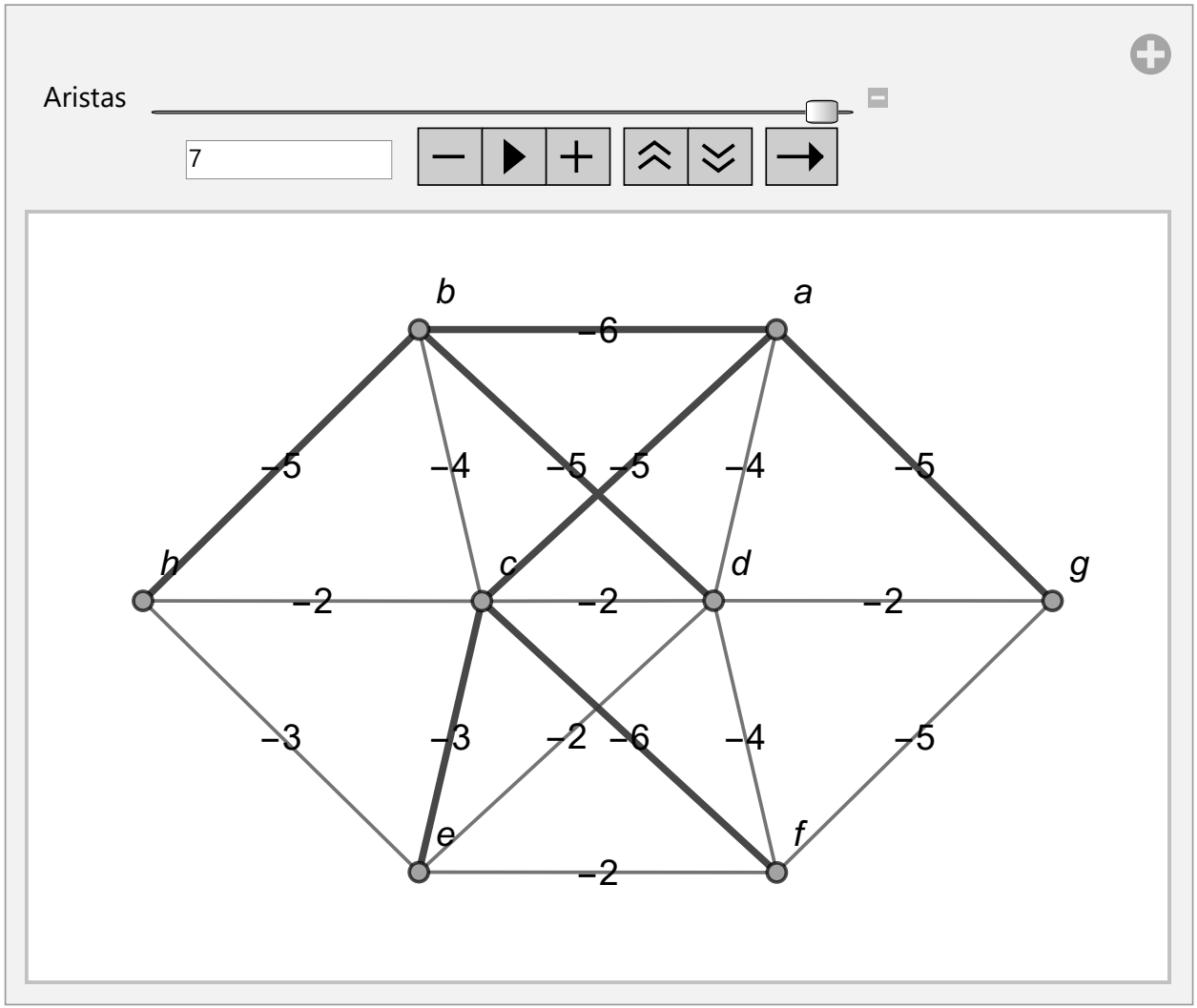


El orden de los nodos es: {b, a, g, e, c, f, h, d}

T={{b,a}, {b,h}, {b,d}, {a,g}, {a,c}, {c,f}, {c,e}}

Peso: -35

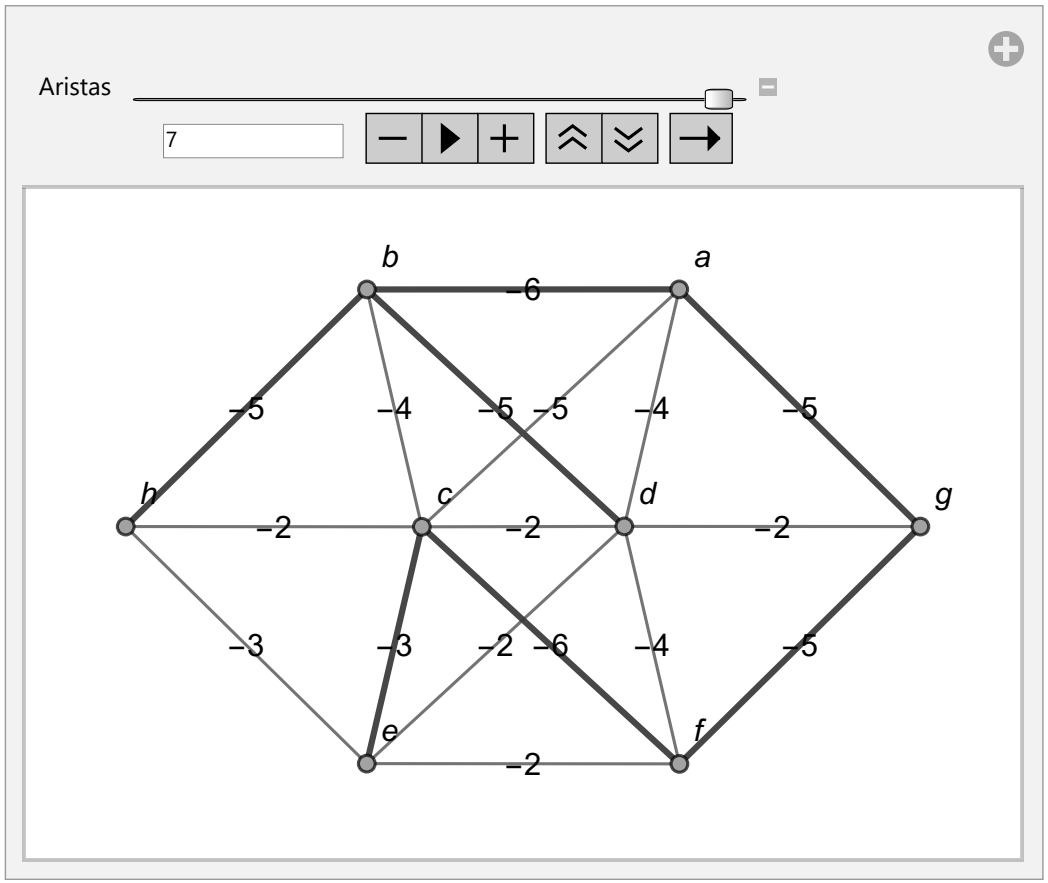
Solución del ejemplo 6.23



Solución del ejemplo 6.23

T={a●–●b, c●–●f, a●–●g, f●–●g, b●–●d, b●–●h, c●–●e}

Peso: -35



Solución del ejemplo 6.23

Se concluye que el peso máximo de cualquier árbol generador en el grafo original es 35. Se insta al alumno a correr el código anterior, utilizando la opción `all -> True` en las sentencias `Prim` y `Kruskal`.

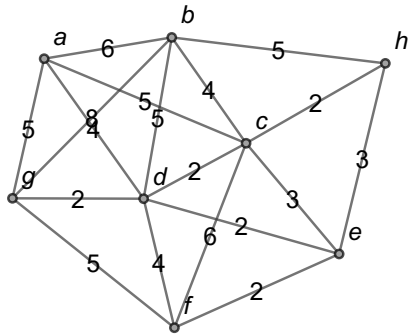
Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/File-159.zip

A continuación se socializa con el estudiante un interesante documento con un formato computable, que facilita la aplicación simultánea de los algoritmos "buscar primero a lo ancho", "buscar primero a lo largo", *Prim* y *Kruskal*, especificando: las aristas del grafo conexo, los pesos asociados a sus lados y el orden de interés para los vértices del grafo. En la figura 6, se comparte un ejemplo de la salida que provee el *CDF* citado, sobre un grafo conexo y ponderado.

Árboles generadores

Aristas del grafo	{{c, e}, {d, e}, {d, a}, {c, d}, {e, h}, {c, h}, {c, f}, {a, b}, {a, c}, {b, c}, {b, d}, {b, h}, {d, f}, {e, f}, {f, g}, {a, g}, {g, d}, {g, b}}
Pesos de los lados	{3, 2, 4, 2, 3, 2, 6, 6, 5, 4, 5, 5, 4, 2, 5, 5, 2, 8}
Orden de los nodos	{c, e, h, f, d, b, a, g}



Buscar primero a lo ancho:

{{c, e}, {c, h}, {c, f}, {c, d}, {c, b}, {c, a}, {f, g}}

Buscar primero a lo largo:

{{c, e}, {e, h}, {h, b}, {d, b}, {f, d}, {f, g}, {a, g}}

Prim: {{{c, h}, {c, d}, {d, e}, {e, f}, {d, g}, {c, b}, {d, a}}, 18}

Kruskal: {{{e, f}, {c, d}, {c, h}, {g, d}, {d, e}, {b, c}, {d, a}}, 18}

Autor: Enrique Vílchez Quesada
Escuela de Informática | Universidad Nacional de Costa Rica

Figura: Funcionamiento del *CDF* "árboles generadores"

CDF: árboles generadores sobre un grafo conexo y ponderado

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/CDFs/Arboles_generadores.cdf.zip

Cuaderno interactivo sobre los contenidos del capítulo

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Cuadernos/Arboles.pdf.rar

Quiz interactivo de repaso sobre los contenidos del capítulo

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/Quiz_arboles.rar

Webmix sobre el empleo de los comandos del paquete **VilCretas** estudiados en el capítulo

Abra un sitio web

`https://www.symbaloo.com/mix/vilcretasarboles`

¡Recuerde resolver los ejercicios asignados!

Descargue un archivo

https://www.escinf.una.ac.cr/discretas/Archivos/Arboles/Exercises.zip

enrique.vilchez.quesada@una.cr
http://www.escinf.una.ac.cr/discretas