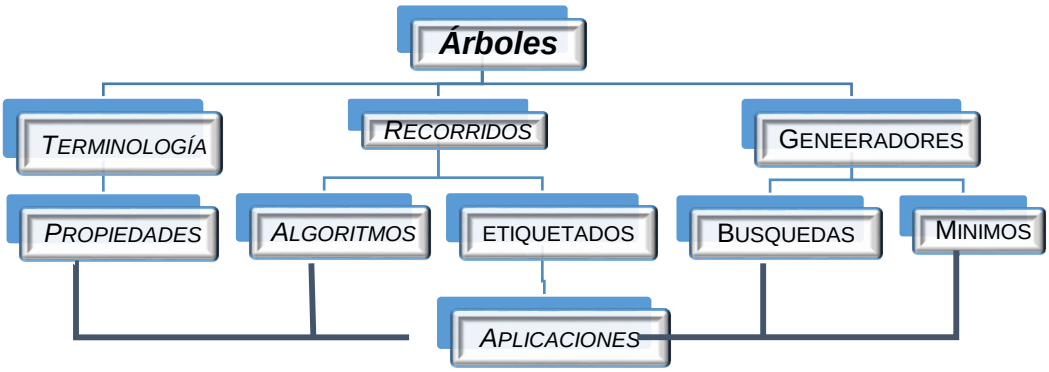
ÁRBOLES


Árboles
TERMINOLOGÍA
RECORRIDOS
GENEERADORES
PROPIEDADES
ALGORITMOS
ETIQUETADOS
BUSQUEDAS
MINIMOS
APLICACIONES

OBJETIVOS

- ✓ Brindar los principios básicos para poder estudiar y realizar aplicaciones del tema árboles.
- ✓ Reconocer los diferentes tipos de árboles, como también sus propiedades y diferentes aplicaciones.
- ✓ Determinar las características que debe poseer un grafo para ser árbol y obtener un árbol a partir de un grafo.
- ✓ Resolver problemas de informática haciendo uso del modelo que brinda el tema árboles.
- ✓ Aplicar la estructura de árbol para la organización y procesamiento de la información.

Árboles

1.- Introducción

Un grafo conexo que no posea ciclos, es un árbol; llamados así porque tales grafos se asemejan a los árboles. Por ejemplo los árboles genealógicos son grafos en los que se presentan relaciones de parentesco. Se utilizan los vértices para los miembros de una familia y las aristas para representar las relaciones entre padres e hijos.

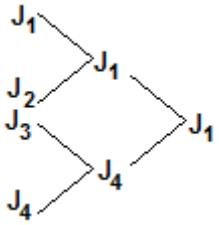
Los árboles empezaron a utilizarse en 1857, cuando el matemático inglés Arthur Cayley los utilizó para contar cierto tipo de componentes químicos. Desde entonces los árboles se han empleado para resolver problemas en una gran variedad de disciplinas, tales como problemas de probabilidad, problemas de toma de decisiones, análisis de la conexión de un grafo, búsqueda de ciclos hamiltonianos para resolver el problema del viajante, problemas de optimización, etc.

Particularmente en informática los árboles son empleados en un amplio espectro de algoritmos. Por ejemplo, para construir algoritmos eficientes que localizan elementos en una lista, para construir códigos compresores eficientes, para ahorrar costes en la transmisión de datos y en su posterior almacenamiento; como los códigos de Huffman. Los árboles también pueden utilizarse para modelar procedimientos que se llevan a cabo mediante una secuencia de decisiones. Construir estos modelos puede ayudar a determinar la complejidad computacional de los algoritmos basados en una secuencia de instrucciones, como los algoritmos de ordenación.

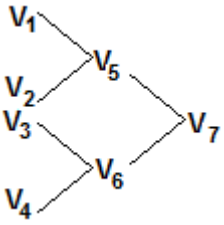
2.- Árboles

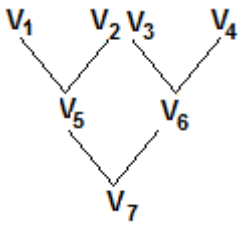
En el tenis existe una competencia que se conoce como *torneo por eliminación sencilla,* cuando un jugador pierde, sale del torneo. Los ganadores siguen jugando hasta que queda sólo una persona: el campeón.

Se podría esquematizar un torneo por eliminación sencilla para cuatro jugadores y observar la trayectoria del campeón; en la siguiente figura.



Si el torneo por eliminación sencilla de la figura anterior es visto como un grafo, se obtiene un árbol. Si se rota este grafo, puede entenderse el porqué de su nombre.





Se pueden dar dos definiciones formales de árboles.

Definición 1:

Un árbol es un grafo no dirigido, conexo y sin ciclos.

Puesto que un árbol no puede contener ciclos, es un grafo acíclico, tampoco puede tener bucles o aristas múltiples; por lo tanto un árbol es necesariamente un grafo simple.

Definición 2:

Un árbol T (libre) es un grafo que satisface lo siguiente: si $v \ y \ w$ *son vértices en T, existe un camino simple de* $v \ a \ w$.

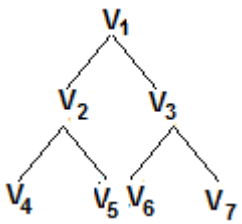
Un árbol con raíz es un árbol en el que un vértice específico se designa como raíz.

Si se designa al ganador como raíz, el torneo por eliminación sencilla del ejemplo es un árbol con raíz. Obsérvese que si $v \ y \ w$ son vértices en este grafo, existe un camino simple único de $v \ a \ w$. Por ejemplo, el camino simple único de $v_6 \ a \ v_2$ se expresa como $v_6, \ v_7, \ v_5, \ v_2$.

Al contrario de los árboles naturales, cuyas raíces se localizan abajo, en la teoría de grafos los árboles con raíces suelen dibujarse con la raíz hacia arriba.

Primero, se coloca la raíz v_1 arriba. Abajo de la raíz y al mismo nivel, se colocan los vértices $v_2 \ y \ v_3$, a los que se puede llegar desde la raíz por un camino simple de longitud 1. Abajo de estos vértices y al mismo nivel se colocan los restantes vértices $v_4, \ v_5, \ v_6 \ y \ v_7$, a los que se llega desde la raíz por caminos simples de longitud 2. Se continúa así hasta dibujar el árbol completo.

Finalmente el árbol del ejemplo que se analiza es el siguiente.

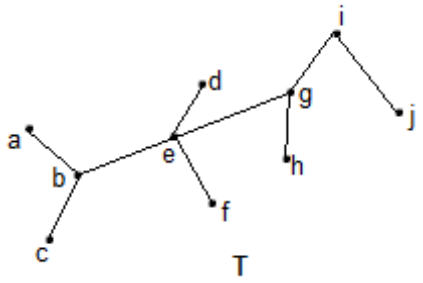


Como el camino simple de la raíz a cualquier vértice dado es único, cada vértice está en un nivel determinado de manera única. El nivel de la raíz es el nivel 0. Se dice que los vértices abajo de la

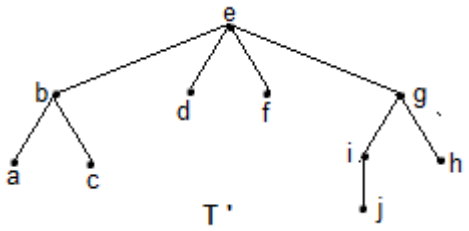
raíz están en el nivel 1, y así sucesivamente. Entonces el *nivel de un vértice* v es la longitud del camino simple de la raíz a dicho vértice. La *altura* de un árbol con raíz es el número máximo de nivel que ocurre.

Para este árbol el vértice v_1 que es la raíz, esta a nivel cero. Los vértices v_2 y v_3 están en el nivel 1 y los vértices $v_4,\ v_5,\ v_6$ y v_7 están en el nivel 2. Por lo tanto la altura del árbol es 2.

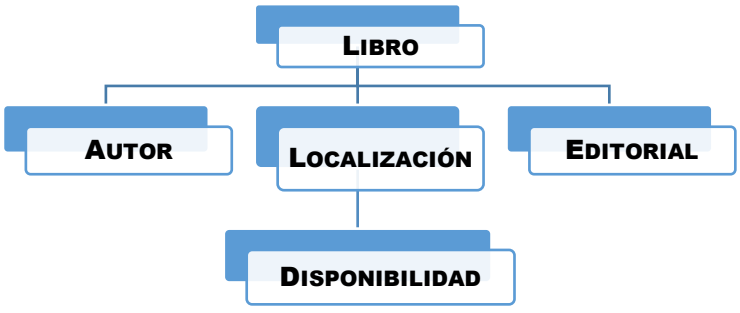
Ejemplo: Sea el siguiente grafo



Si se designa, por ejemplo al vértice e como raíz se obtiene el árbol con raíz T'. En donde se puede observar que los vértices $b,\ d,\ f$ y g están en el nivel 1 mientras que los vértices $a,\ c,\ i$ y h están el nivel 2. El vértice j esta en el nivel 3. La altura de T' es 3.



Otro ejemplo de árbol son los llamados *árboles de definición jerárquica.* Estos árboles se usan para mostrar las relaciones entre los registros de una base de datos. El árbol que se presenta a continuación se puede usar como modelo para establecer una base de datos para mantener los registros de los libros en varias bibliotecas.



2.1.- TERMINOLOGÍA Y CARACTERIZACIÓN DE LOS ÁRBOLES

La terminología de los árboles tiene orígenes botánicos y genealógicos.

Definición:

Sea T un árbol con raíz v_0. Supóngase que x, y, z son vértices en T y que un camino simple en T es $v_0, v_1, \ldots v_n$. Entonces:

a) v_{n-1} es el padre de v_n.

b) $v_0, \ldots v_{n-1}$ son ancestros de v_n.

c) v_n es un hijo de v_{n-1}.

d) Si x es un ancestro de y; y es un descendiente de x.

e) Si x e y son hijos de z, x e y son hermanos.

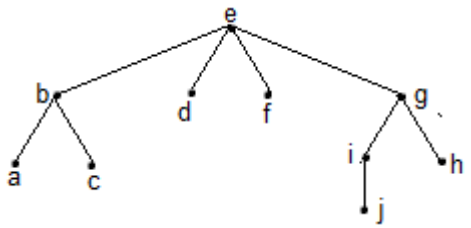
f) Si x no tiene hijos, x es un vértice terminal u hoja.

g) Si x no es un vértice terminal, x es un vértice interno.

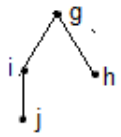
h) El subárbol de T con raíz en x, es el subgrafo del árbol que contiene al vértice x, a todos sus descendientes y a todas las aristas incidentes en dichos descendientes.

i) El nivel de un vértice v es la longitud del camino simple de la raíz a dicho vértice. La altura de un árbol con raíz es el número máximo de nivel que ocurre.

Sea el siguiente árbol con raíz "e".



El vértice g es el padre de los vértices i y h, por lo tanto i y h son hermanos. El vértice j es una hoja cuyos ancestros son i, g, e. Los vértices internos del árbol son b, e, g, e i. El subárbol con raíz en g es el siguiente.



La raíz e tiene nivel 0, mientras que los vértices b, d, f, y g forman el nivel 1. Los vértices a, c, i y h el nivel 2 y solamente el vértice j forma el nivel 3. La altura del árbol es 3.

Los árboles con raíz que cumplen la propiedad de que todos sus vértices internos tienen el mismo número de hijos, se utilizan en múltiples aplicaciones.

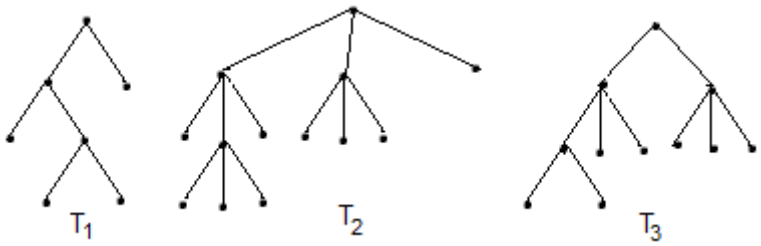
Definición:

Un árbol con raíz se llama m–ario si todos los vértices internos tienen, a lo sumo, m hijos.

El árbol se llama árbol m–ario completo si todo vértice interno tiene exactamente m hijos.

Un árbol m–ario con m = 2 se llama binario.

Sean los siguientes árboles:



T_1 es un árbol binario completo ya que todos sus vértices internos tienen exactamente dos hijos, T_2 es un árbol ternario completo pues todos sus vértices internos tienen exactamente tres hijos. El árbol T_3 no es un árbol completo debido a que algunos de sus vértices internos tienen dos hijos mientras que otros tienen tres.

Un árbol ordenado con raíz es un árbol con raíz en el que los hijos de cada vértice interno están ordenados. Estos árboles se dibujan de modo que los hijos de cada vértice interno se colocan ordenados de izquierda a derecha. Se hará uso de estas ordenaciones, sin hacer mención que se está considerando un árbol ordenado con raíz.

2.2.- PROPIEDADES DE LOS ÁRBOLES

Con frecuencia se necesitan resultados que relacionen los números de vértices y aristas en diferentes tipos de árboles. Para estas ocasiones son de utilidad los siguientes teoremas.

Teorema 1:

Un árbol con n vértices tiene $n-1$ aristas.

Demostración.

Se usará Inducción Matemática para demostrar este teorema. Nótese que para cualquier árbol, se puede elegir una raíz y considerar el árbol con raíz resultante.

PB) Para $n = 1$, un árbol con un vértice no tiene aristas. De allí que el teorema sea cierto para $n = 1$.

PI) La hipótesis inductiva afirma que todo árbol con k vértices tiene $k-1$ aristas, siendo k un entero positivo. Supóngase que un árbol T tiene $k+1$ vértices y v es una hoja de T (que debe existir ya que el árbol es finito) y sea w el padre de v. Si eliminamos de T tanto el vértice v como la arista que une a v con w se obtiene un árbol T' de k vértices, puesto que el grafo es conexo y no tiene ciclos. Por la hipótesis inductiva, T' tiene $k-1$ aristas. De ahí se deduce que T tiene k

aristas ya que tiene una más que T', la arista que conecta a v con w. Esto completa el paso inductivo.

Conclusión, un árbol con n vértices tiene $n-1$ aristas.

El número de vértices de un árbol m–ario completo con un número prefijado de vértices internos está determinado, como muestra el siguiente teorema. Como en el caso anterior se denotará con n al número de vértices de un árbol.

Teorema 2:

Un árbol m–ario completo con i vértices internos tiene $n = mi + 1$ vértices

Demostración.

Todo vértice, excepto la raíz, es hijo de algún vértice interno. Puesto que cada uno de los i vértices internos tiene m hijos, hay $m.i$ vértices en el árbol, diferentes a la raíz. Por lo tanto el árbol tiene un total de $n = m.i + 1$ vértices.

Supóngase que T es un árbol m–ario completo. Sea i el número de vértices internos y l el número de hojas del árbol. Una vez que algunos de los enteros n, i ó l es conocido, las otras dos magnitudes quedan determinadas. En el siguiente teorema se establece como calcular las otras dos cantidades a partir de la que ya se conoce.

Teorema 3:

Un árbol m–ario completo con:

1.- n vértices, tiene $i = \frac{n-1}{m}$ vértices internos y $l = \frac{(m-1)n+1}{m}$ hojas.

2.- i vértices internos, tiene $n = m.i + 1$ vértices y $l = (m-1)i + 1$ hojas.

3.- l hojas tiene, $n = \frac{m.l-1}{m-1}$ vértices e $i = \frac{l-1}{m-1}$ vértices internos.

Demostración.

Sea n el número de vértices, i el número de vértices internos y l el número de hojas. Los tres apartados del teorema se pueden demostrar usando la igualdad establecida en el teorema 2, $n = m.i + 1$, junto con la igualdad $n = l + i$ que es cierta porque cada vértice es bien una hoja o un vértice interno.

Se despeja i de $n = m.i + 1$, entonces $i = \frac{n-1}{m}$ Sustituyendo esta expresión en la igualdad $n = l + i$ se tiene $n = l + \frac{n-1}{m}$ de donde resulta $l = n - \frac{n-1}{m}$ Operando esta expresión se obtiene que $l = \frac{(m-1)n+1}{m}$ Quedando demostrado el primer apartado del teorema.

Para el apartado dos del teorema ya se tiene, por el teorema 2 que $n = m.i + 1$ En esta expresión se sustituye la igualdad $n = l + i$, obteniéndose $l + i = m.i + 1$ de donde se obtiene que $l = (m-1)i + 1$.

Para el apartado tres se despeja i de $n = l + i$, o sea que $i = n - l$. Se reemplaza esta expresión en $n = m.i + 1$. Obteniéndose $n = m(n - l) + 1$ de donde al despejar n se obtiene $n = \frac{m.l-1}{m-1}$. Ahora en la expresión $i = n - l$ se reemplaza $n = m.i + 1$ *obteniéndose* $i = m.i + 1 - l$. *Se despeja* i y se obtiene $i = \frac{l-1}{m-1}$. De esta manera queda demostrado el apartado tres y en consecuencia también el teorema.

Supóngase que alguien inicia una cadena de cartas. A cada persona que recibe cada una de esas cartas se le pide que la envíe a otras cuatro. Algunas personas lo hacen pero otras no envían ninguna carta. ¿Cuántas personas han leído la carta, incluyendo a la primera persona, si nadie recibe más de una carta y si la cadena finaliza después de que 100 personas que han visto la carta no hayan enviado ninguna? ¿Cuántas personas enviaron la carta?

La cadena de cartas puede ser representada por medio de un árbol 4–ario, en donde los vértices internos (i) se corresponden con aquellas personas que enviaron la carta y las hojas (l) con aquellas otras que no la enviaron.

Puesto que 100 personas no enviaron la carta el número de hojas de este árbol es 100 ($l = 100$). De allí por el apartado 3 del teorema 3 se tiene que el número de personas que han enviado la carta es $n = \frac{4.100-1}{4-1} = \frac{399}{3} = 133$. También se tiene que el número de vértices internos es $i = \frac{100-1}{4-1} = 33$, o sea que 33 personas enviaron la carta.

3.- RECORRIDOS EN ÁRBOLES

Los árboles ordenados con raíz se pueden utilizar tanto para almacenar información como para representar varios tipos de expresiones; tales como expresiones aritméticas que involucran números, variables y operadores. Por lo tanto es necesario determinar procedimientos que permitan visitar cada uno de los vértices de dichos árboles para acceder a los datos.

3.1.- ALGORITMOS DE RECORRIDOS

Se describirán tres algoritmos de recorrido de un árbol, que se utilizan con más frecuencia. El nombre del recorrido indica el orden en el que se coloca el padre en relación a sus hijos. Los tipos de recorrido son: en orden primero, en orden segundo y en orden final.

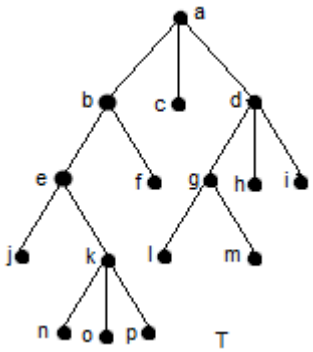
Definición:

Sea T un árbol ordenado con raíz r. Si T consta solo de r, entonces r es el recorrido en orden primero de T. En otro caso, supóngase que $T_1, \ldots T_n$ *son los subárboles de r listados de izquierda a derecha en T. El recorrido en orden primero comienza visitando r, continua recorriendo* T_1 *en orden primero, luego* T_2 *en orden primero y así sucesivamente hasta recorrer* T_n *en orden primero.*

En el recorrido en orden primero (padre, hijo izquierdo, demás hijos) se toma el padre luego el hijo izquierdo y al final los demás hijos. Es decir que se comienza por la raíz, después se sigue por el vértice de la izquierda, si este vértice tiene hijos se sigue por el de la izquierda hasta llegar a la hoja. Si esta hoja tiene hermanos se toma el que está más cercano a ella (más a la izquierda). Después de que se termina con la rama izquierda, se continúa con la rama más cercana a ella y así sucesivamente hasta terminar con el recorrido de todo el árbol.

Ejemplo:

Sea T un árbol ordenado con raíz.



El recorrido en orden primero es: a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i.

El algoritmo del recorrido en orden primero de un árbol ordenado, se puede expresar en forma recursiva de la siguiente manera.

ALGORITMO: Recorrido en orden primero

procedure ordenprimero (T: árbol ordenado con raíz)
r:= raíz de T
enumerar r
for cada hijo c de r de izquierda a derecha
begin
    T(c) := subárbol de raíz c
    ordenprimero (T (c))
end

Definición:

Sea T un árbol ordenado con raíz r. Si T consta solo de r, entonces r es el recorrido en orden segundo de T. En otro caso, supóngase que $T_1, \ . \ . \ . \ T_n$ *son los subárboles de r listados de izquierda a derecha en T. El recorrido en orden segundo comienza recorriendo*

T_1 en orden segundo y continúa visitando r, a continuación recorre T_2 en orden segundo y así sucesivamente hasta recorrer T_n en orden segundo.

En el recorrido en orden segundo (hijo izquierdo, padre, demás hijos) se toma el hijo izquierdo, segundo el padre y al final los demás hijos. Es decir que se comienza por la hoja que se encuentra más a la izquierda del árbol, después se regresa al padre y posteriormente a todos los hermanos, después se regresa al padre de esta rama y con las ramas de éste (tomando siempre la que está más a la izquierda) y así sucesivamente hasta terminar con el recorrido de todo el árbol.

Continuando con el árbol T del ejemplo anterior. El recorrido en orden segundo de dicho árbol será: j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i.

El algoritmo del recorrido en orden segundo de un árbol ordenado, se puede expresar en forma recursiva de la siguiente manera.

ALGORITMO: Recorrido en orden segundo

```
procedure ordensegundo (T: árbol ordenado con raíz)
r:= raíz de T
if r es una hoja then enumerar r
else
begin
        l:= primer hijo de r de izquierda a derecha
        T ( l ) := subárbol de raíz l
        ordensegundo ( T ( l ) )
        listar r
for cada hijo c de r excepto para l y de izquierda a derecha
        T( c ) := subárbol de raíz c
        ordensegundo(T ( c ) )
end
```

Definición:

Sea T un árbol ordenado con raíz r. Si T consta solo de r, entonces r es el recorrido en orden final de T. En otro caso, supóngase que $T_1, \ldots T_n$ son los subárboles de r listados de izquierda a derecha en T. El recorrido en orden final comienza recorriendo T_1 en orden final, luego recorre T_2 en orden final y así sucesivamente hasta recorrer T_n en orden final y termina visitando la raíz r.

En el recorrido en orden final (hijo izquierdo, demás hijos, padre) se toma primero el hijo izquierdo, después los demás hijos y al final el padre. Se comienza en la hoja que se encuentra más a la izquierda en el árbol, después se continúa con los hermanos, sí estos tienen hijos

primeramente recorre los hijos y hasta el final el padre, dando preferencia a los hijos de la izquierda y hasta el final el padre. En este tipo de recorrido lo último que se recorre es la raíz, ya que tienen preferencia los hijos sobre el padre.

Continuando con el árbol T del ejemplo anterior. El recorrido en orden final de dicho árbol será: j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a.

ALGORITMO: Recorrido en orden final

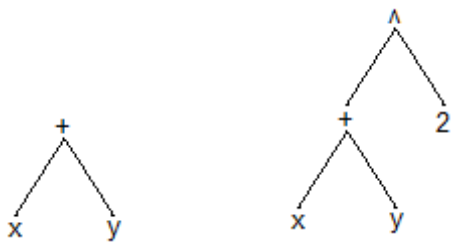
procedure ordenfinal (T: árbol ordenado con raíz)
r:= raíz de T
for cada hijo c de r de izquierda a derecha
begin
    T(c) := subárbol de raíz c
    ordenfinal (T (c))
end
listar r

3.2.- RECORRIDO EN ÁRBOLES ETIQUETADOS.

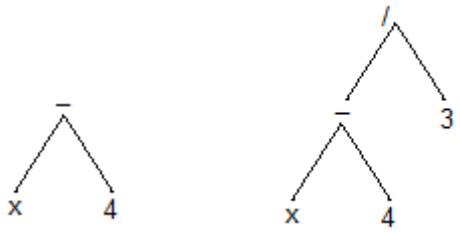
Se pueden representar expresiones complejas, tales como fórmulas proposicionales, combinaciones de conjuntos y expresiones aritméticas mediante árboles ordenados con raíz. Por ejemplo se puede representar una expresión aritmética que implique las operaciones: suma, resta, producto, cociente y potenciación; en este caso los vértices internos del árbol representan operaciones mientras que las hojas representan variables o números.

Por ejemplo ¿Cuál es el árbol ordenado con raíz que representa la siguiente expresión $(x+y)^2 + \frac{x-4}{3}$?

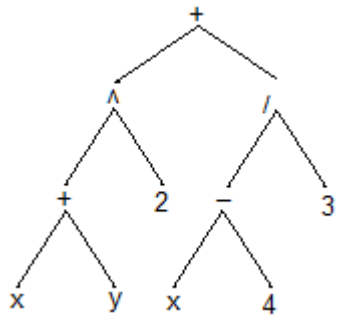
El árbol binario completo para esta expresión se puede construir de abajo hacia arriba. Primero se construye un subárbol para la expresión $(x+y)$ que posteriormente se unirá a otro subárbol mayor que representa la expresión $(x+y)^2$.



También se construye un subárbol para $(x-4)$ que luego se incorporará a otro subárbol mayor que representa a $\frac{x-4}{3}$



Finalmente el subárbol que representa a $(x+y)^2$ y el subárbol que representa a $\frac{x-4}{3}$ se combinan para formar el árbol ordenado con raíz que representa a toda la expresión.



¿Cuál es el recorrido de este árbol en orden primero?

El recorrido en orden primero será: **+ ^ + x y 2 / – x 4 3**

En el recorrido en orden primero de una expresión, un operador binario, como por ejemplo + precede a sus dos operandos; por lo tanto se puede evaluar una expresión de este tipo trabajando de derecha a izquierda. Cuando se encuentra un operador se realiza el cálculo correspondiente con los dos operandos que se encuentran inmediatamente a su derecha. Siempre que se realiza una operación, el resultado se considera como un operando.

Evaluar la expresión del recorrido en orden primero, para x = 7 e y = 1.

+ ^ + x y 2 / – x 4 3

+ ^ + 7 1 2 / – 7 4 3

+ ^ + 7 1 2 / 3 3

+ ^ + 7 1 2 1

+ ^ 8 2 1

+ 64 1

65

¿Cuál es el recorrido del árbol anterior pero en orden final?

El recorrido en orden final será: **x y + 2 ^x 4 – 3 / +**

En el recorrido en orden final de una expresión, los operadores binarios van detrás de sus dos operandos. Por lo tanto se puede evaluar una expresión de este tipo trabajando de izquierda a derecha, realizando las operaciones siempre que un operador siga a dos operandos. Tras realizar la operación, el resultado se convierte en un nuevo operando.

Evaluar la expresión del recorrido en orden final, para x = 7 e y = 1.

x y + 2 ^ x 4 − 3 / +

7 1 + 2 ^ 7 4 − 3 / +

8 2 ^ 3 3 / +

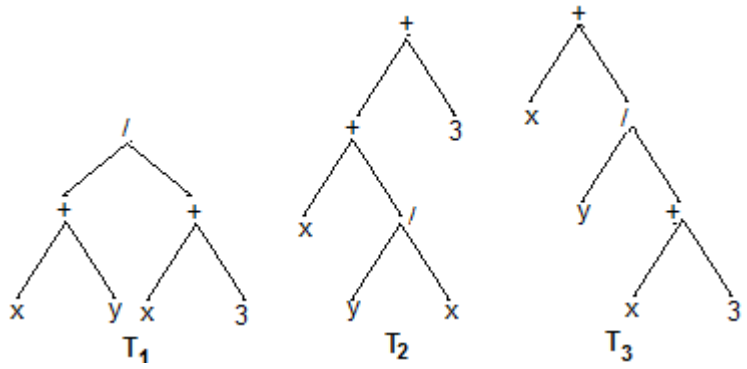
64 1 +

65

El recorrido en orden segundo de un árbol binario que representa una expresión reproduce la expresión original con los elementos y las operaciones en el orden en que aparecían inicialmente, excepto para los operadores unarios que aparecen inmediatamente después de sus operandos.

Por ejemplo sean las expresiones $E_1 = \frac{x+y}{x+3}$, $E_2 = \left(x + \frac{y}{x}\right) + 3$, $E_3 = x + \frac{y}{x+3}$

Los árboles T_1, T_2 y T_3 que representan, respectivamente, a estas expresiones son los siguientes.



Compruébese que el recorrido en orden segundo para los tres árboles produce exactamente la misma expresión: x + y / x + 3. Para evitar la ambigüedad, es necesario incluir paréntesis en el recorrido en orden segundo siempre que se encuentre una operación.

De esta manera la expresión que produce el recorrido en orden segundo del árbol T_1 es $(x + y)/(x + 3)$, mientras que para T_2 es $(x + (y/x)) + 3$. Finalmente para el árbol T_3 se obtiene la siguiente expresión: $x + (y/(x + 3))$.

Los árboles con raíz se pueden utilizar para representar otro tipo de expresiones, como las fórmulas proposicionales y las combinaciones entre conjuntos. En estos casos pueden aparecer operadores unarios como la negación en el caso de las fórmulas. Para representar este tipo de operador y a su operando se utiliza un vértice para el operador y un hijo para su argumento.

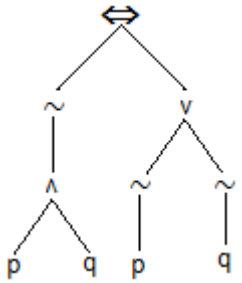
Ejemplo.

Obtener el árbol ordenado con raíz que representa a la siguiente fórmula proposicional $\sim(p \wedge q) \Leftrightarrow (\sim p \vee \sim q)$.

Primero se construye el subárbol que representa a la expresión $(p \wedge q)$ que se unirá a un subárbol mayor que representa a la expresión $\sim(p \wedge q)$.

Luego se construyen los subárboles de $\sim p \;\; y \;\; \sim q$ (la negación es un operador unario) y se unen al subárbol que representa a la expresión $(\sim p \vee \sim q)$.

Finalmente los dos subárboles obtenidos se utilizan en la construcción del árbol con raíz de la expresión.



El recorrido de este árbol en orden primero es $\Leftrightarrow \sim \wedge \; p \; q \;\; \vee \sim \;\; p \;\; \sim \;\; q$, mientras que el recorrido en orden final es: $p \;\; q \;\; \wedge \sim \;\; p \;\; \sim \;\; q \;\;\; \sim \vee \Leftrightarrow$. Los recorridos en orden primero y en orden final se utilizan frecuentemente en informática, ya que su escritura no es ambigua y se pueden evaluar fácilmente en una sola lectura. Estas expresiones son especialmente útiles en la construcción de compiladores.

4.- ÁRBOLES GENERADORES

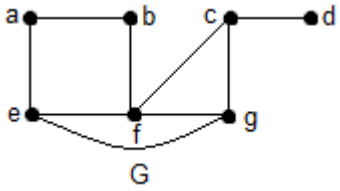
Algunas aplicaciones necesitan algoritmos para decidir cuando un grafo tiene ó no, una determinada propiedad, por ejemplo, la de ser ó no conexo ó para encontrar todos los posibles caminos hamiltonianos ó para contar el número de veces que aparece una cierta estructura. Por lo general estas aplicaciones se basan en un árbol generador.

Definición:

Sea G un grafo simple. Un árbol generador o abarcador de G es un subgrafo de G que es un árbol y contiene todos los vértices de G.

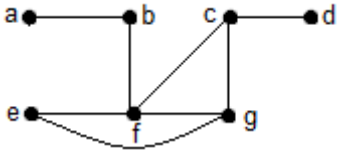
Un grafo simple que admite un árbol generador es necesariamente conexo, ya que existe un camino en el árbol generador entre dos vértices cualesquiera. El reciproco también es válido, es decir, todo grafo simple conexo tiene un árbol generador.

Ejemplo: Obtener un árbol generador del siguiente grafo simple G.

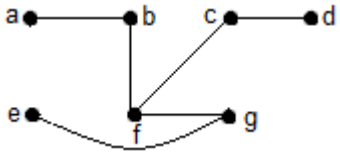


G

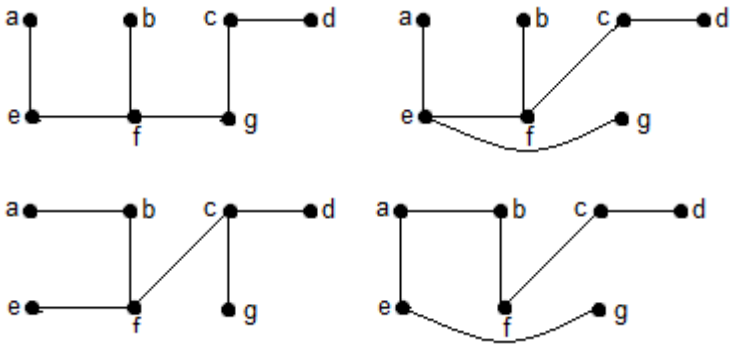
El grafo G es conexo, pero no es un árbol porque contiene ciclos. Quítese la arista $\{a,\ e\}$. De este modo se elimina un ciclo y el subgrafo resultante todavía es conexo y contiene todos los vértices de G.



Seguidamente quítese la arista $\{e,\ f\}$ para eliminar otro ciclo y quítese la arista $\{c,\ g\}$ para obtener un grafo simple y sin circuitos. Este último subgrafo es un árbol generador puesto que es un árbol que contiene todos los vértices de G.



Este no es el único árbol generador de G. Por ejemplo cada uno de los siguientes árboles también es un árbol generador de G.



Existen muchas formas de construir árboles generadores. De hecho, para un mismo grafo existen diferentes árboles generadores; como afirma el teorema de Cayley "*un mismo grafo puede tener hasta* n^{n-2} *árboles generadores diferentes*" (n es el número de vértice del grafo).

En general existen técnicas para contar el número de árboles generadores de un grafo cualquiera, sin embargo las formas más comunes de construir árboles generadores tienen que ver con procedimientos de búsqueda, denominados búsqueda en profundidad y búsqueda en anchura.

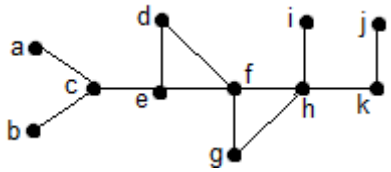
4.1.- PROCEDIMIENTO DE BÚSQUEDA EN PROFUNDIDAD

Partiendo de la raíz v del árbol, el procedimiento consiste en elegir como vértice activo la propia raíz y construir un árbol parcial W de la siguiente manera.

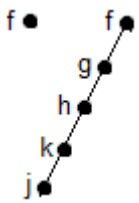
Denomínese x al vértice activo, siempre que el vértice activo tenga nuevos vértices adyacentes, elíjase uno de ellos, por ejemplo y. Añádase la arista $\{x, y\}$ a W. Avanzar a y luego sustitúyase x por y como nuevo vértice activo. Si no hay nuevos vértices adyacentes a x, retrocédase al vértice que originalmente condujo hasta x. En algún momento del proceso se volverá de nuevo a v sin posibilidad de añadir nuevos vértices, entonces $W = T$. Con este procedimiento cada arista del árbol T es recorrida dos veces, una para avanzar y otra para retroceder.

Si v es un vértice del grafo G y T es el árbol generador construido mediante el procedimiento de búsqueda en profundidad entonces T es un árbol generador de G que contiene a v.

Ejemplo: Utilizar la búsqueda en profundidad para obtener un árbol generador del siguiente grafo G.

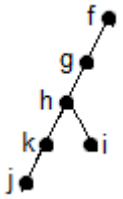


Elíjase de modo arbitrario un vértice inicial f que es el vértice activo. Como el vértice activo f tiene vértices adyacentes se elije g, (se podría haber elegido d) ahora g es el vértice activo que también tiene vértices adyacentes. Se elige h (vértice activo), a partir de allí se elige k (se podría haber elegido i), por último se elige a partir del vértice activo k, el vértice adyacente a él j. Es decir que al árbol W se le agregaron a partir de la raíz f las siguientes aristas.



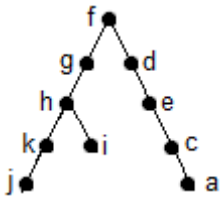
Se retrocede desde j hasta k (vértice activo) pero como este no tiene vértices adyacentes que no hayan sido utilizado se retrocede hasta h (vértice activo). A partir del vértice activo se puede elegir únicamente i, pues g ya fue utilizado.

Es decir que se aumentó a W la arista $\{ h, i \}$.

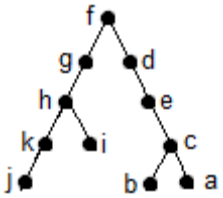


A partir del vértice activo i se retrocede hasta h (vértice activo) que no tiene vértices adyacentes que no hayan sido utilizados, se retrocede hasta g en donde ocurre lo mismo y finalmente el vértice activo es f.

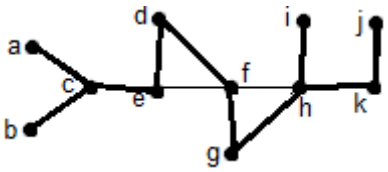
A partir de f se puede elegir el vértice d, a partir de allí el vértice e y de allí el c, terminando en el vértice a (vértice activo). Es decir que al árbol W se le agregaron las siguientes aristas.



Finalmente a partir del vértice activo a se retrocede hasta el vértice c desde donde se puede elegir únicamente b. Vale decir que se aumento al árbol, la arista $\{ c, b \}$ y por lo tanto $W = T$. Es decir que el árbol generador para el grafo G es el siguiente.



Las aristas de un grafo seleccionadas en la búsqueda en profundidad se llaman *aristas del árbol*. Todas las demás aristas del grafo tienen que conectar un vértice con un antecesor o un descendiente de ese vértice en el árbol. A estas aristas se las llama *aristas de retroceso*. A continuación se destacan, con trazo grueso, las aristas del árbol utilizadas por la búsqueda en profundidad del ejemplo anterior. Las aristas de retroceso son $\{ e, f \}$ y $\{ f, h \}$.



El pseudocódigo para la búsqueda en profundidad es el siguiente.

ALGORITMO: Busqueda en profundidad

procedure Busq_Prof (G: grafo conexo de vértices $v_1,\ v_2,\ .\ .\ .\ v_n$)

T:= árbol que consta sólo del vértice v_1

visita (v_1)

procedure visita (v: vértice de G)

for cada vértice w adyacente a v y que no esté en T

begin

añadir el vértice w y la arista $\{ v, w \}$ a T

visita (w)

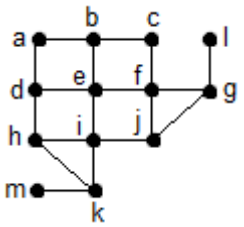
end

El algoritmo construye un árbol generador de un grafo G que tiene por conjunto de vértices a v_1, v_2, . . . v_n. Primeramente se selecciona el vértice v_1 como raíz, iniciando el proceso al asignar al árbol T justamente ese único vértice. En cada paso se añade un nuevo vértice al árbol T junto con una arista que lo conecta con un vértice de T y se explora desde ese nuevo vértice. Nótese que al terminar el algoritmo, T no contiene ningún circuito, ya que no se añaden aristas a vértices que ya están en el árbol. Por otro lado, en todas las fases del proceso de construcción, T es un grafo conexo. Puesto que G es conexo, todo vértice de G se utiliza en alguna fase del algoritmo y se añade al árbol. De esto se concluye que T es un árbol generador de G.

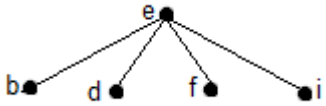
4.2.- PROCEDIMIENTO DE BÚSQUEDA EN ANCHURA

También se puede obtener un árbol generador de un grafo simple mediante la *búsqueda en anchura o por niveles*. Se elige un vértice arbitrario como raíz y se añaden todas las aristas incidentes en ese vértice. Los nuevos vértices añadidos en esta fase forman los vértices del nivel 1 del árbol generador. Se ordenan los vértices del nivel 1 con un orden cualquiera. Para cada vértice del nivel 1, visitados en orden, se añaden todos los vértices incidentes con él, siempre que no formen un ciclo. Se ordenan los hijos de los vértices del nivel 1 con un orden cualquiera, generando así los vértices del nivel 2 del árbol. Se continúa de esta manera hasta que se hayan añadido todos los vértices al árbol. Este procedimiento termina, ya que los grafos tienen un número finito de aristas. Lo que se obtiene es un árbol generador porque se obtuvo un subgrafo que es un árbol que contiene a todos los vértices del grafo.

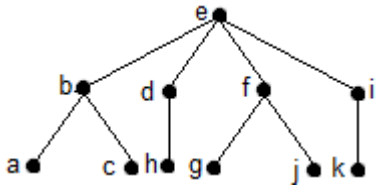
Ejemplo: Utilizar la búsqueda en anchura para obtener un árbol generador del siguiente grafo G.



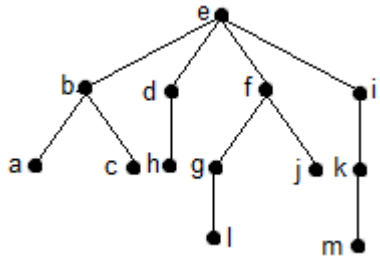
Se elige un vértice como raíz, e. Se añaden todas las aristas incidentes con todos los vértices adyacentes a e, esto es, las aristas $\{e, b\}, \{e, d\}, \{e, f\}, \{e, i\}$. Los cuatros vértices extremos de las aristas anteriores, se añaden al nivel 1 del árbol.



Se añaden las aristas que conectan los vértices del nivel 1 con vértices que aún no están en el árbol. Por lo tanto se añaden $\{b, a\}, \{b, c\}, \{d, h\}, \{f, g\}, \{f, j\}, \{i, k\}$. Los nuevos vértices a, c, h, j, g, k forman el nivel 2.



Se añaden las aristas que unen los vértices del nivel 2 con vértices adyacentes que no están en aún en el árbol. Estas aristas son $\{g, l\}, \{k, m\}$. Los vértices l y m forman el nivel 3 del árbol.



En el algoritmo, en pseudocódigo para la búsqueda en anchura, se considera que los vértices del grafo G están ordenados y etiquetados por ejemplo de la siguiente manera: $v_1,\ v_2,\ \ldots\ v_n$.
Se utiliza el término *proceso* para describir el procedimiento de añadir nuevos vértices y las correspondientes aristas al árbol adyacentes al vértice procesado mientras no se generen ciclos.

ALGORITMO: Busqueda en anchura

procedure Busq_Anch (G: grafo conexo de vértices $v_1,\ v_2,\ \ldots\ v_n$)
T:= árbol que consta sólo del vértice v_1
L:= lista vacía
poner v_1 en la lista L de los vértices no procesados
while L no esté vacía
begin
borrar el primer vértice v de L

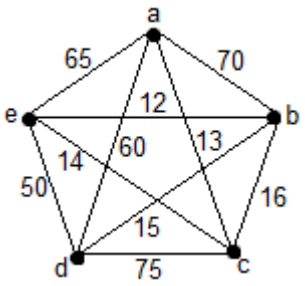
```
        for cada vértice w adyacente a v
            if w no está en L y no está en T then
            begin
                añadir el vértice w al final de la lista L
                añadir el vértice w y la arista { v, w } a T
            end
end
```

5.- ÁRBOL GENERADOR MÍNIMO

Existen situaciones, como puede ser la de conectar una serie de ciudades mediante un sistema de autopistas ó conectar una serie de edificios mediante una red de telecomunicaciones, en donde el problema no es exactamente saber si existe un árbol generador, sino más bien saber cuál es el árbol generador más económico, siendo que las redes en ambos ejemplos tienen un determinado costo. Esto conduce a los conceptos de árbol ponderado y árbol generador mínimo. Supóngase que una compañía planea construir una red de comunicaciones para conectar sus cinco centros informáticos. Cualquier pareja de estos centros debe estar enlazada mediante una línea telefónica. ¿Qué enlaces deberían realizarse para asegurar que hay un camino entre cualquier par de centros de modo que el costo total de la red sea el menor posible?

Se puede modelar este problema usando un grafo ponderado donde los vértices representan los centros informáticos, las aristas representan posibles líneas de comunicaciones y los pesos de las aristas son los costos de construir las líneas asociadas a las aristas.

Está claro que para comunicar dos edificios no es necesario un enlace directo, pues se puede mandar un mensaje de uno a otro en forma indirecta. Por ejemplo enviar un mensaje de a hacia b y de b hacia c, en lugar de enviarlo en forma directa de a hacia c.



Definición:

Sea G= (V, E) un grafo. Si se asocia a cada arista de G un valor numérico, w(e) que se denomina peso de la arista e, a la función $w: E \to R^+$ *se la denomina función peso; entonces se dice que G y w constituyen un grafo ponderado.*

Claramente, se puede asociar esta definición a la red de telecomunicaciones del problema que se viene analizando. Aquí los pesos de las aristas vienen dados por sus respectivos costos. El objetivo es conseguir que ese costo sea el mínimo, lo que corresponde a un árbol generador T de G cuyo peso $w(T) = \sum_{e \in T} w(e)$ sea el menor posible.

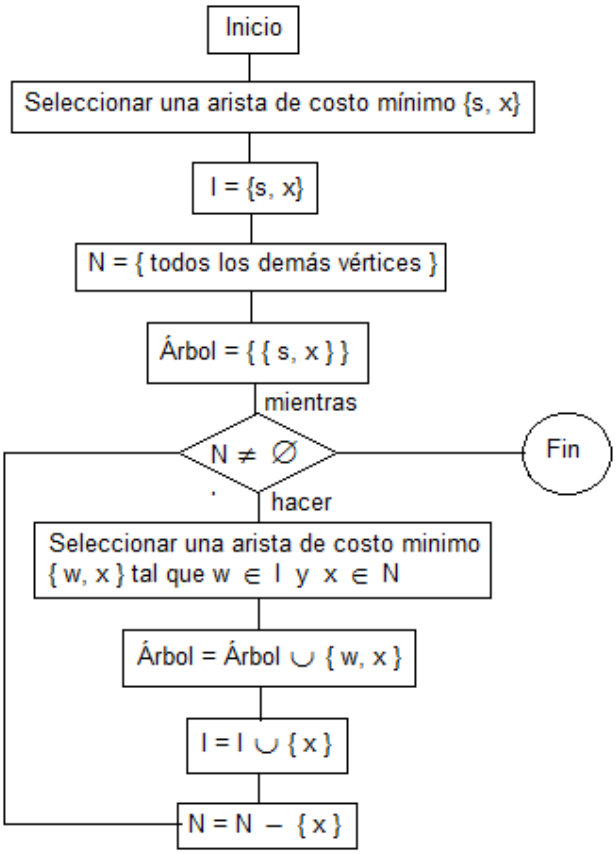
Definición:

Un árbol generador mínimo de un grafo ponderado, es un árbol generador tal que la suma de los pesos de sus aristas es la mínima posible de entre todos los árboles generadores.

Existen dos algoritmos que conducen de una manera directa a un árbol generador mínimo de un grafo G de n vértices.

El primer algoritmo fue propuesto por Robert Prim en 1957, aunque las ideas básicas son anteriores. En este método los vértices se dividen en dos conjuntos: vértices integrados (I) –que son los que forman parte del árbol generador mínimo– y los vértices no integrados (N). El conjunto Árbol contiene todas las aristas que se van integrando en cada iteración. En cada paso se agrega un vértice en el conjunto I mientras que el conjunto N es disminuido en uno.

Se puede expresar el *Algoritmo de Prim* mediante el siguiente diagrama de flujo.



Nótese que la elección de una arista para añadirla en cualquiera de los pasos del algoritmo no está determinada cuando hay más de una con el mismo peso satisfaciendo los criterios. Por lo

tanto si en el momento de seleccionar la arista de peso mínimo se opta por otra de igual peso, entonces el árbol generador mínimo es diferente.

El segundo algoritmo para construir un árbol generador de peso mínimo fue descubierto por Joseph Kruskal en 1956, aunque las ideas que se utiliza fueron descritas con antelación a esa fecha.

Para ejecutar el *Algoritmo de Kruskal* se elige una arista del grafo de entre las que tienen menor peso.

Se añaden paulatinamente las aristas con menor peso siempre que estas no formen un ciclo con otras ya incorporadas. El proceso termina cuando se han seleccionado $n - 1$ aristas.

El pseudocódigo de este algoritmo es el siguiente.

ALGORITMO: Algoritmo de Kruskal

procedure Kruskal (G: grafo ponderado, conexo y no dirigido de n vértices)
T:= grafo vacío
for i:= 1 **to** n – 1
begin
    e:= una arista de peso mínimo de G que no forme un ciclo cuando se añada al grafo T
    T:= T con e añadida
end

Tanto el Algoritmo de Prim como el de Kruskal son algoritmos voraces[1], pudiéndose demostrar que ambos proporcionan un árbol generador de peso mínimo para cualquier grafo ponderado conexo.

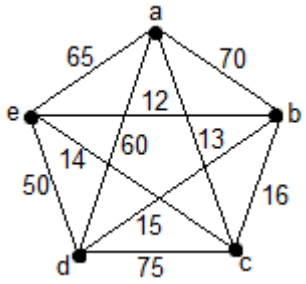
Otra cuestión que comparten ambos algoritmos es el hecho de que si las aristas no están ordenadas, hay más de una posible elección para las aristas que se añaden en los distintos pasos de los algoritmos.

Adviértase que la diferencia entre los algoritmos es la siguiente: en el Algoritmo de Prim las aristas de peso mínimo elegidas deben ser incidentes con algunos de los vértices del árbol ya construido y no deben forman un ciclo con las ya elegidas, en el Algoritmo de Kruskal, en cambio, las aristas de peso mínimo que se eligen no pueden forman un ciclo con las ya elegidas, pero no deben ser necesariamente incidentes con vértices del árbol.

Se poseen las condiciones para resolver el problema planteado al iniciar este apartado.

[1] Un algoritmo voraz es un procedimiento que realiza una elección óptima en cada uno de sus pasos. El hecho de que se optimice en cada paso no garantiza que la solución final sean una solución óptima. Sin embargo, esta situación no ocurre para estos dos algoritmos.

Ejemplo: Encontrar los árboles generadores mínimos que se obtienen usando los algoritmos de Prim y de Kruskal en el siguiente grafo ponderado.



En esencia; el algoritmo de Prim consiste en repetir el siguiente paso hasta que el conjunto Árbol tenga $n-1$ aristas (n es el número de vértices del grafo dado): añadir a Árbol la arista de menor peso entre un vértice de Árbol y un vértice que no esté en él.

Se empieza por la arista de menor peso, en este caso, $Árbol = \{\{b,e\}\}$. Se añade a Árbol la arista $\{e,c\}$ de peso 14, con lo que $Árbol = \{\{b,e\},\ \{e,c\}\}$ de peso 26. En el siguiente paso se añade la arista $\{c,a\}$; con lo que $Árbol = \{\{b,e\},\ \{e,c\},\ \{c,a\}\}$ de peso 39. Finalmente se añade la arista $\{b,d\}$; con lo que termina el algoritmo pues, se añadieron cuatro aristas sin formar un ciclo.

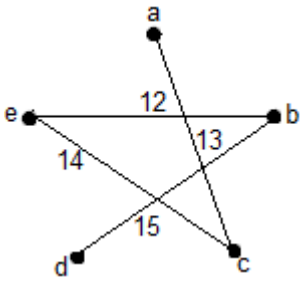
Entonces $Árbol = \{\{b,e\},\ \{e,c\},\ \{c,a\},\ \{b,d\}\}$ de peso 54.

Básicamente, el algoritmo de Kruskal consiste en repetir el siguiente paso hasta que el conjunto T tenga $n-1$ aristas: añadir a T la arista con menor peso que no forme un ciclo con las aristas que ya están en T.

Se inicia con la arista de menor peso, en este caso $\{b,e\}$, por lo tanto $T = \{\{b,e\}\}$ de peso 12. Se añade a T la arista de menor peso, en este caso $\{a,c\}$. Ahora se tiene $T = \{\{b,e\},\ \{a,c\}\}$ de peso 25. En el siguiente paso se añade $\{c,e\}$ con lo que el conjunto $T = \{\{b,e\},\ \{a,c\},\ \{c,e\}\}$ de peso 39. Finalmente se añade la arista $\{b,d\}$ con lo que se añadieron cuatro aristas a T sin formar un ciclo.

Entonces $T = \{\{b,e\},\ \{e,c\},\ \{c,a\},\ \{b,d\}\}$ de peso 54.

El árbol generador mínimo obtenido por los dos algoritmos es el siguiente.



Nótese que en el algoritmo de Prim se parte de un árbol y se van obteniendo árboles en las sucesivas iteraciones. Sin embargo en el de Kruskal no es necesariamente así, se van obteniendo distintos bloques –que no tienen por qué ser árboles– ya que sólo al final se puede asegurar que se obtiene un árbol. Nuevamente, en ambos algoritmos, cuando haya varias opciones para añadir nuevas aristas, cualquiera de las opciones puede ser elegida. Aunque es inmediato comprobar que ambos algoritmos conducen a árboles generadores, es algo más complicado demostrar que estos son además, mínimos.

6.- APLICACIONES DE LOS ÁRBOLES

6.1.- CÓDIGOS INSTANTÁNEOS

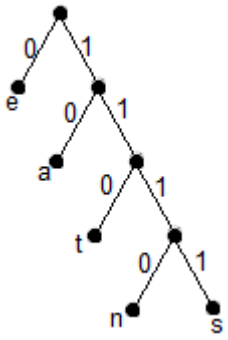
Considérese el siguiente problema. Codificar las letras de un alfabeto mediante cadenas de bits –sin distinguir entre mayúsculas y minúsculas– de tal manera de maximizar el ahorro de espacio en la memoria y minimizar el tiempo de transmisión de datos.

Una solución es codificar las letras con cadenas de bits de longitudes variables, es decir, las letras que aparezcan con mayor frecuencia deberán codificarse utilizando cadenas de bits más cortas, mientras que las letras menos frecuentes se codificarán mediante cadenas más largas.

Cuando se codifica con cadenas de longitud variables se debe establecer algún método para determinar cuando comienza y cuando termina cada cadena de bits. Por ejemplo, si p se codificara por 0, e por 1 y s por 01, entonces la cadena 0101 podría corresponder a pes, spe, ss o pepe.

Una manera de asegurar de que ninguna cadena de bits se corresponde con más de una secuencia de caracteres consiste en codificar las letras de manera que la cadena de bits asociada a una letra nunca aparezca al principio de la cadena de bits de otra letra. Los códigos con esta propiedad se llaman códigos instantáneos. Por ejemplo la codificación de p como 0, e como 10 y ss como 11 es un código prefijo. Toda palabra puede reconstruirse a partir de la única cadena de bits que codifica los caracteres que le corresponde. Por ejemplo, la cadena 10110 es la codificación de esp, nótese que el 1 inicial no representa un carácter pero 10 representa la letra e (y no puede ser el inicio de la cadena de bits de ninguna otra letra). Por lo tanto el siguiente 1 no representa un carácter sino que 11 representa la letra s. El bit final, 0, representa la letra p.

Un código instantáneo puede representarse utilizando un árbol binario donde los caracteres son las etiquetas de las hojas del árbol. Las aristas del árbol están etiquetadas de modo que a la arista que va al hijo izquierdo se le asigna 0 y a la que va al hijo derecho se le asigna 1. La cadena de bits que codifica cada carácter es la sucesión de las etiquetas de las aristas del único camino de la raíz a la hoja que tiene a este carácter como etiqueta. Por ejemplo en el siguiente árbol se codificó los caracteres e, a, t, n y s.



Los mismos árboles que representan códigos instantáneos se pueden utilizar para decodificar cadenas de bits. Por ejemplo considérese la palabra codificada, según el árbol anterior, como 1 1 1 1 0 1 1 1 0 1 0. Esta cadena de bits se puede decodificar comenzando por la raíz, utilizando la sucesión de bits para formar un camino desde la raíz que termina al alcanzar una hoja. Cada bit con valor 0 hace que se descienda por la arista que lleva al hijo izquierdo del último vértice del camino y cada 1 se corresponde con el hijo derecho de dicho vértice. Por lo tanto la subcadena 1 1 1 1 se corresponde con la letra s. Comenzando con el quinto bit, se alcanza la hoja luego de ir a la derecha una vez y luego a la izquierda, o sea que la cadena 10 se corresponde con el carácter a. Se inicia, nuevamente, desde la raíz y a partir del séptimo bit llegándose a una hoja luego de usar la subcadena 1 1 1 0 que se corresponde con el carácter n. Finalmente el resto de la cadena es 10 que codifica el carácter a. En consecuencia la palabra codificada es $sana$.

Se puede construir un código instantáneo a partir de cualquier árbol binario donde la arista izquierda de cada vértice interno esté etiquetada con un 0 y la arista derecha con un 1 y en el que las hojas estén etiquetadas con caracteres. Los caracteres se codifican mediante las cadenas de bits construidas utilizando las etiquetas de las aristas que hay en el único camino desde la raíz hasta las hojas.

Códigos de Huffman. David Huffman en un artículo que escribió en 1.951 cuando aún era estudiante de doctorado, desarrolló un algoritmo que toma como datos de entrada las frecuencias (probabilidad de aparición) de símbolos de una cadena y devuelve un código instantáneo que codifica la cadena utilizando la menor cantidad de bits, de entre todos los posibles códigos instantáneos binarios para este conjunto de símbolos. Este algoritmo conocido como *codificación de Huffman* supone que ya se sabe cuántas veces aparece cada símbolo en la cadena, así que se puede calcular la frecuencia de cada símbolo dividiendo el número de apariciones del símbolo entre la longitud de la cadena. La codificación de Huffman es un algoritmo esencial en la comprensión de datos, que es el área de conocimiento que se dedica a reducir el número de bits necesarios para representar la información. También se utiliza para comprimir cadenas de bits que representan textos, archivos de audios y de imágenes.

6.2.- Árboles de Juego

Los árboles se pueden emplear para analizar ciertos tipos de juegos como ser: ta–te–ti, nim, damas ó ajedrez. En estos juegos los jugadores realizan sus movimientos por turnos. Cada jugador conoce los movimientos efectuados por el contrario y no hay ninguna componente de azar en el juego. Estos juegos se modelan mediante *árboles de juegos*, en donde los vértices representan los estados o posiciones en el desarrollo del juego y las aristas representan los movimientos permitidos entre dos estados cualesquiera.

La raíz representa el estado inicial y usualmente se conviene en representar los vértices de los niveles impares por rectángulos y los de los niveles pares por círculos. Cuando el juego está en una posición representada por un vértice de nivel impar, es turno del primer jugador, mientras que si está en una de nivel par es el turno del segundo jugador.

Las hojas de un árbol de juegos representan las posiciones finales de un juego. Se asigna un valor a cada hoja indicando la puntuación que obtiene el primer jugador si el juego termina en la posición representada por esa hoja.

Para los juegos que son ganador–perdedor, se etiqueta un vértice terminal representado mediante un círculo con un 1 para indicar una partida ganada por el primer jugador y se etiqueta un vértice terminal representado por un rectángulo con un –1 para indicar que gana el segundo jugador. Nótese que para este tipo de juego, la asignación de valores a los vértices terminales significa que cuanto mayor sea el valor, tanto mejor será el resultado final del juego para el primer jugador.

El juego nim. En una versión del juego *nim*, al inicio de la partida hay un cierto número de fichas distribuidas en filas. Dos jugadores hacen sus movimientos por turnos; un movimiento válido consiste en retirar una o más fichas de una de las filas, (sin quitar ninguna de las fichas restantes). Un jugador pierde si no puede realizar ningún movimiento válido. (Otro modo de ver este juego es que el jugador que retira la última ficha pierde, puesto que la posición sin fichas no está permitida).

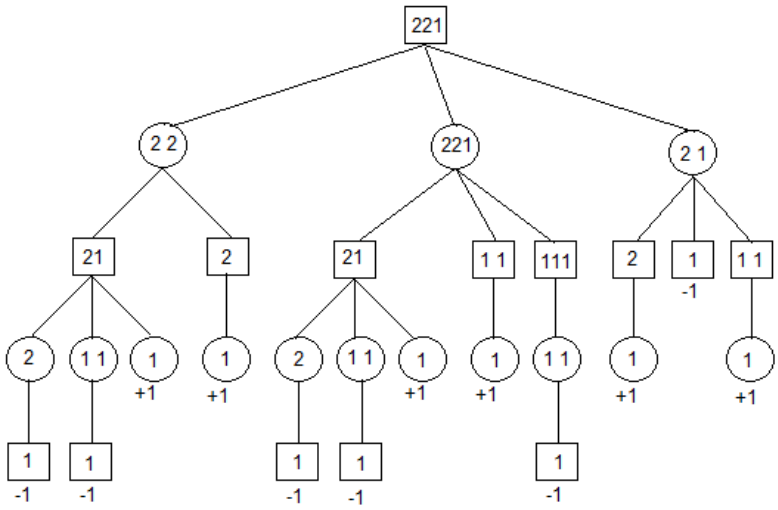
El árbol que se muestra a continuación representa esta versión del juego que comienza con una posición inicial de tres filas con dos, dos y una ficha respectivamente. Se representa cada posición con una lista no ordenada del número de fichas en las distintas filas (el orden de las fichas es irrelevantes).

El movimiento inicial del primer jugador puede llevar a tres posiciones posibles ya que puede retirar a lo sumo dos fichas.

Si retira una ficha de la fila que tiene una sola ficha, deja dos filas con dos fichas cada una; si retira una ficha de alguna de las dos filas que tienen dos fichas, deja una fila con dos fichas y dos filas

con una ficha cada una; de retirar dos fichas de alguna de las dos filas posibles, deja dos filas con una y dos fichas respectivamente. Cuando queda una fila con una ficha, no es posible realizar ningún movimiento válido, por lo tanto esas posiciones son terminales.

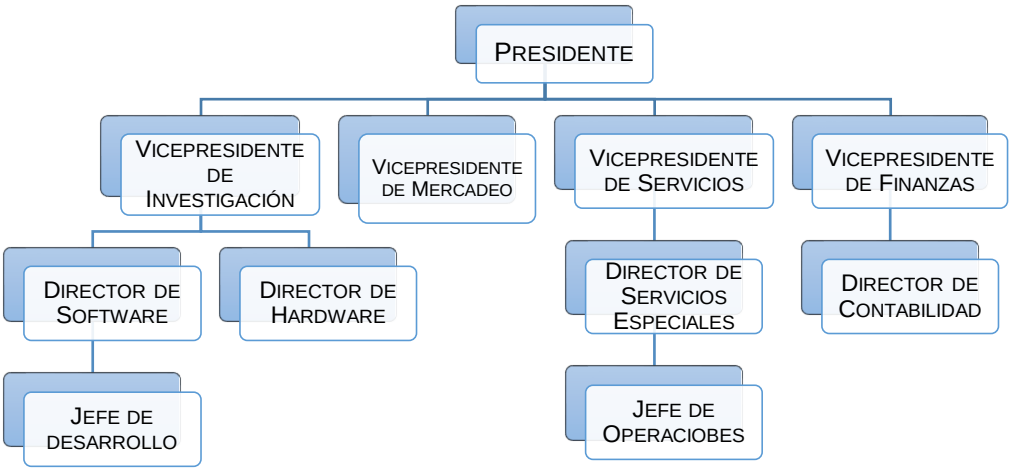
Puesto que el juego del nim es del tipo ganador–perdedor, se etiquetan los vértices terminales con +1 cuando se representa que el primer jugador gana y con –1 cuando lo hace el segundo.



6.3.- ÁRBOLES COMO MODELOS

Representación de organizaciones. La estructura de una organización se puede modelar utilizando un árbol con raíz. Cada vértice de este árbol indica un puesto en la organización. Una arista de uno de los vértices a otro indica que la persona representada por el vértice inicial es jefe (directo) de la persona representada por el vértice final. En el grafo mostrado a continuación es un ejemplo de árbol de este tipo.

En la organización representada por este árbol, el vicepresidente de servicios es jefe directo del director de servicios especiales pero también del jefe de operaciones. El presidente de la organización trabaja directamente con los cuatro vicepresidentes.

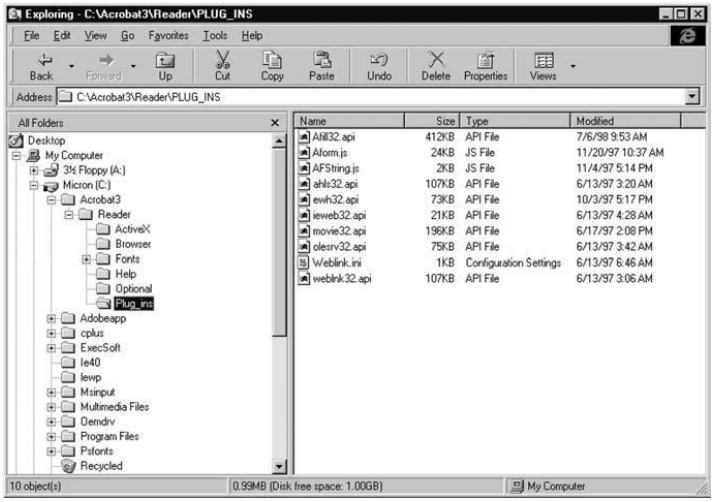


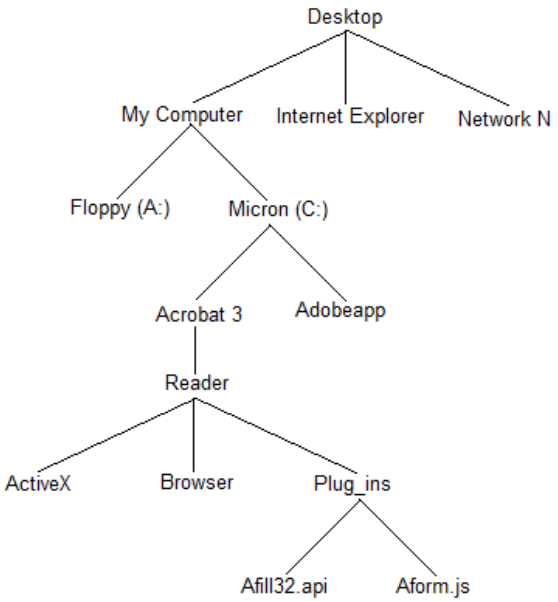
Sistema de archivos de computadoras. En la memoria de una computadora los archivos se organizan en directorios. Un directorio puede contener tanto subdirectorios como archivos. La raíz de un directorio contiene el sistema de archivo completo. De esta manera un sistema de archivos puede representarse mediante un árbol con raíz, donde la raíz representa el directorio raíz, los vértices internos representan los subdirectorios y las hojas representan los archivos comunes o subdirectorios vacíos.

A continuación se muestra el explorador de Windows con el despliegue de carpetas a la izquierda y los archivos a la derecha en una computadora en particular y luego la misma estructura como un árbol con raíz.

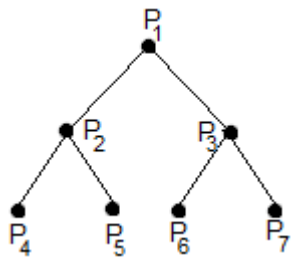
La raíz se llama Desktop. Por debajo de ella están My Computer, Internet Explorer, y otros (My Computer es lo único desplegado). Abajo de My Computer están Floppy (A:), Micron (C:) y otros que no se muestran.

Abajo de Plug_ins, que está resaltado, están los archivos Afill32.api, Aform.js y otros, que aparecen a la derecha en la primera figura.





Procesadores en paralelos conectados en árbol. Una red conectada en árbol es otra manera de conectar computadoras. El grafo que representa este tipo de red es un árbol binario completo. Una red de este tipo conecta $n = 2^k - 1$ procesadores, donde k es un entero positivo. Un procesador representado por un vértice v que no es raíz o una hoja, tiene tres conexiones bidireccionales –una al procesador representado por el padre de v y dos a los procesadores representados por los hijos de v–. A continuación se muestra una red conectada en árbol con siete procesadores.



Mediante un sencillo ejemplo se ilustrará como se puede usar una red conectada en árbol para el cálculo en paralelo. Se muestra en particular como se utilizan los procesadores del árbol anterior para sumar ocho números en tres pasos. En el primer paso se suma $x_1\ y\ x_2$ con P_4; $x_3\ y\ x_4$ con P_5; $x_5\ y\ x_6$ con P_6 y $x_7\ y\ x_8$ con P_7. En el segundo paso se suma $x_1 + x_2\ \ y\ \ x_3 + x_4$ con P_2 y $x_5 + x_6\ \ y\ \ x_7 + x_8$ con P_3 finalmente en el tercer paso se suma $x_1 + x_2 + x_3 + x_4\ y\ \ x_5 + x_6 + x_7 + x_8$ con P_1.

Los tres pasos utilizados para sumar ocho números mejoran los siete pasos necesarios para sumarlos en serie, donde cada paso consiste en sumar un número con la suma acumulada de dos números situados previamente en la lista.