

Ingeniería En Sistemas Computacionales

Estructura de Datos

Grupo: 3J2

Unidad 4: “Árboles y Grafos”

Integrantes:

* Karla Beatriz Martínez Solís
* Oscar Iván García Rivera
* Eduardo Daniel Romero Enríquez

Catedrático: Ing. Luis Bernardo Ballesteros Barradas

H. Ciudad Y Puerto De Veracruz, Veracruz

12/11/2014

ÍNDICE

Introducción 3

4.1 Concepto de Árbol 4

4.2 Clasificación de Árboles 7

4.3 Operaciones Básicas sobre Árboles Binarios 12

4.4 Aplicaciones de Árboles 16

4.5 Árboles Balanceados 20

4.6 Grafos 22

4.7 Terminología de Grafos 23

4.8 Operaciones Básicas sobre Grafos 24

Conclusión 26

Referencias 27

INTRODUCCIÓN

El presente trabajo trata sobre las estructuras de datos no lineales: árboles y grafos. En ambos tipos de estructuras se profundizara sobre su definición y las operaciones que podremos realizar con ellos.

Los árboles en estructura de datos tienen ciertos conceptos parecidos a los que encontramos en un árbol en la vida real, por ejemplo, raíz, rama, hoja, anchura, altura, lo que puede ayudar a comprender mejor su uso y funciones. En el trabajo se hablaran lo los tipos de árboles sin embargo es necesario mencionar que el más usado en computación es el binario, el cual tiene 0, 1 o 2 descendientes como máximo.

Los grafos guardan cierto parecido con los árboles, la diferencia es que los grafos poseen ciclos. En el trabajo se hablara también de la terminología así como de la teoría de grafos.

4.1 CONCEPTO DE ÁRBOL

Un árbol es una estructura de datos ramificada (no lineal) que puede representarse como un conjunto de nodos enlazados entre sí por medio de ramas. La información contenida en un nodo puede ser de cualquier tipo simple o estructura de datos. Los árboles permiten modelar diversas entidades del mundo real tales como, por ejemplo, el índice de un libro, la clasificación del reino animal, el árbol genealógico de un apellido, etc.

Otra definición: Un árbol es una estructura de datos, que puede definirse de forma recursiva como:

Una estructura vacía o un elemento o clave de información (nodo) más un número finito de estructuras tipo árbol, disjuntos, llamados subárboles. Si dicho número de estructuras es inferior o igual a 2, se tiene un árbol binario.

Es, por tanto, una estructura no secuencial.

Otra definición más nos da el árbol como un tipo de grafo (ver [grafos](http://www.algoritmia.net/articles.php?key=grafos)): un árbol es un grafo a cíclico, conexo y no dirigido. Es decir, es un grafo no dirigido en el que existe exactamente un camino entre todo par de nodos. Esta definición permite implementar un árbol y sus operaciones empleando las representaciones que se utilizan para los grafos. Sin embargo, en esta sección no se tratará esta implementación.

La figura 4.1 muestra un ejemplo de estructura en árbol (la numeración de los nodos es arbitraria). Se entiende por “topología” de un árbol a su representación geométrica.

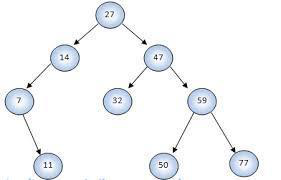


Figura 4.1. Ejemplo de árbol.

Una definición formal es la siguiente:

Un árbol es una estructura de datos base que cumple una de estas dos condiciones:

* Es una estructura vacía, o
* Es un nodo de tipo base que tiene de 0 a N subárboles disjuntos entre sí.

Al nodo base, que debe ser único, se le denomina raíz y se establece el convenio de representarlo gráficamente en la parte superior.

En un árbol se representa una relación jerárquica a partir del nodo raíz en sentido vertical descendente, definiendo niveles1. El nivel del nodo raíz es 1.

Desde la raíz se puede llegar a cualquier nodo progresando por las ramas y atravesando los sucesivos niveles estableciendo así un camino. En la figura 4.1. El nodo 7 está a nivel 3 y la secuencia del nodo 14, constituye un (sub)camino.

Se dice que un nodo es antecesor de otro cuando ambos forman parte de un camino y el primero se encuentra en un nivel superior (numeración más baja) al del segundo (numeración más alta).

La relación entre dos nodos separados de forma inmediata por una rama se denomina padre/hijo. En el ejemplo de la figura 4.1., el nodo 7 es hijo del nodo 14 y, recíprocamente, el nodo 14 es padre del nodo 7. En un árbol un padre puede tener varios hijos pero un hijo solo puede tener un padre.

Se denomina grado al número de hijos de un nodo. Por ejemplo, en la figura 4.1. El nodo 47 tiene grado 2 y el nodo 32 tiene grado 0.

Se dice que un nodo es hoja cuando no tiene descendientes (grado 0).

Se establecen los siguientes atributos para un árbol:

* Altura / profundidad / nivel: La mayor altura / profundidad / nivel de sus nodos. La altura del árbol de la figura 4.1. es 47 (la alcanzan sus nodos 50 y 77).
* Amplitud / Anchura: El número de nodos del nivel más poblado. En el ejemplo, 7 (nivel 3).
* Grado: el mayor de los grados de los nodos. En el ejemplo, 3 (nodos 27 y 47).

Finalmente, indicar que se dice que un árbol es completo cuando todos sus nodos (excepto las hojas) tienen el mismo grado y los diferentes niveles están poblados por completo. A veces resulta necesario completar un árbol añadiéndole nodos especiales.

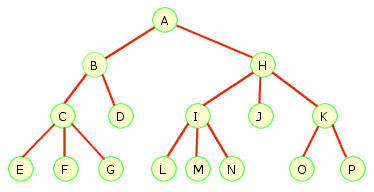
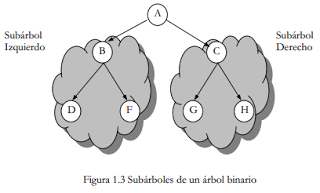


Fig. 4.1a Ejemplo Árbol Completo

4.2 CLASIFICACIÓN DE ÁRBOLES

En ciencias de la computación, un árbol binario es una estructura de datos en la cual cada nodo siempre tiene un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno. Usos comunes de los árboles binarios son los arboles binarios de búsqueda, los montículos binarios y codificación de Huffman.

Un árbol ordenado es aquel en el que las ramas  de los nodos del árbol están  ordenadas. Los árboles ordenados de grado 2 son de especial interés puesto que  representan una de las estructuras de  datos más importantes en computación, conocidas como  árboles binarios. En un árbol binario cada nodo puede tener como máximo dos subárboles y  siempre es necesario distinguir entre el subárbol izquierdo y el subárbol derecho.

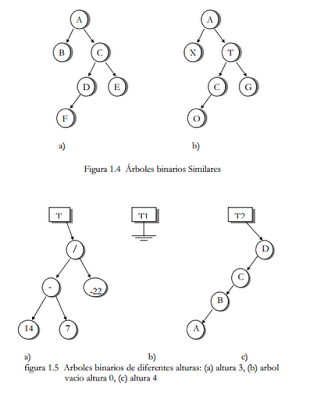
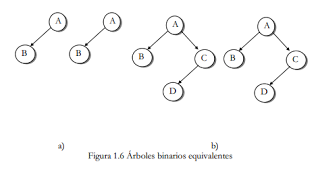


Formalmente podemos definir un árbol binario de tipo T como una estructura homogénea que es la concatenación de un elemento de tipo T, llamada raíz, con dos árboles binarios disjuntos. Una forma  particular de árbol binario puede ser la estructura vacía.

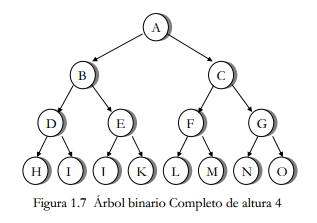
Los árboles binarios  se clasifican en cuatro tipos que son: distintos, similares, equivalentes y completos. Cuando dos  árboles binarios se dice que son similares si tiene la misma estructura y son equivalentes  si son similares  y contienen la misma información. En caso contrario  se dice que  estos árboles son distintos. Un árbol binario está equilibrado  si la altura de los dos subárboles  de cada nodo del árbol  se diferencia en una unidad como máximo.



El procedimiento de árboles binarios equilibrados es más sencillo que los árboles no equilibrados.

Se define un árbol  completo  (lleno)  como un  árbol  en  el  que  todos  sus  nodos, excepto los del ultimo nivel, tienen dos hijos ; el subárbol izquierdo y el subárbol derecho.



Se puede calcular el número de nodos de un árbol binario completo de altura h, aplicando la siguiente formula.



Un árbol binario t es un conjunto finito de nodos tales que

1. t es vacío
2. t consiste de un nodo r, llamado raíz de t y dos árboles disyuntos t1 y t2 llamados subárboles izquierdo y subárbol derecho.

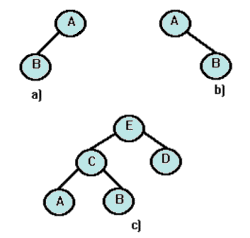


Figura 1.8. Árboles binarios a) subárbol derecho vacío.

b) subárbol izquierdo vacío. c) un árbol binario.

Dos árboles son distintos cuando sus estructuras son diferentes. En la figura 6.2 todos los árboles son distintos.

Dos árboles son similares cuando sus estructuras son idénticas pero la información de los nodos es diferente.

Dos árboles son equivalentes cuando son similares y además tienen la misma información.

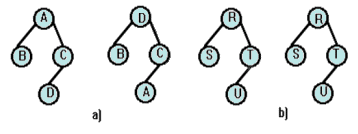


Figura 1.9. a) árboles distintos. b) árboles equivalentes

Árboles Binarios Completos

Un árbol binario es completo cuando todos los nodos de un árbol, excepto los del ultimo nivel tienen dos hijos: subárbol izquierdo y subárbol derecho.

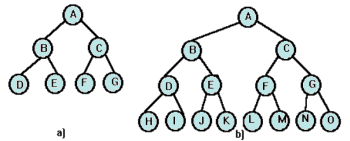


Figura 1.10. Árboles Binarios

El número de nodos en un árbol binario completo de altura h es:

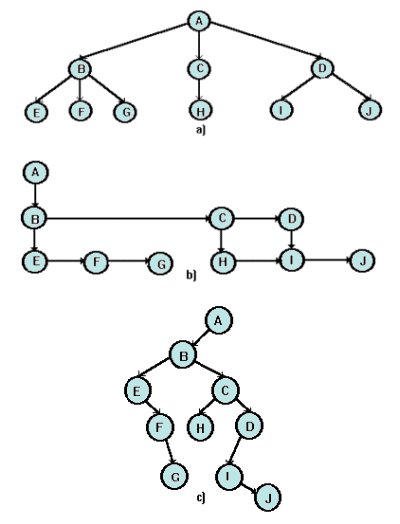
Nodos: 2h - 1.

Conversión de Árboles No Binarios a Binarios

Debido a que los árboles binarios son estructuras muy estudiadas, por esta razón en muchas aplicaciones se convierten árboles no binarios en árboles binarios. Para realizar la conversión se siguen las siguientes reglas:

1. Los nodos hermanos se enlazan en forma horizontal.
2. El nodo raíz se enlaza en forma vertical con el subárbol izquierdo.
3. Se gira la estructura resultante 450.

Ejemplo:



4.3 OPERACIONES BÁSICAS SOBRE ÁRBOLES BINARIOS

Como en toda estructura de datos hay dos operaciones básicas, inserción y eliminación.

**Inserción**

El procedimiento de inserción en un árbol binario de búsqueda es muy sencillo, únicamente hay que tener cuidado de no romper la estructura ni el orden del árbol.

Cuando se inserta un nuevo nodo en el árbol hay que tener en cuenta que cada nodo no puede tener más de dos hijos, por esta razón si un nodo ya tiene 2 hijos, el nuevo nodo nunca se podrá insertar como su hijo. Con esta restricción nos aseguramos mantener la estructura del árbol, pero aún nos falta mantener el orden.

Para localizar el lugar adecuado del árbol donde insertar el nuevo nodo se realizan comparaciones entre los nodos del árbol y el elemento a insertar. El primer nodo que se compara es la raíz, si el nuevo nodo es menor que la raíz, la búsqueda prosigue por el nodo izquierdo de éste. Si el nuevo nodo fuese mayor, la búsqueda seguiría por el hijo derecho de la raíz.

Este procedimiento es recursivo, y su condición de parada es llegar a un nodo que no tenga hijo en la rama por la que la búsqueda debería seguir. En este caso el nuevo nodo se inserta en ese hueco, como su nuevo hijo.

Vamos a verlo con un ejemplo sobre el siguiente árbol:

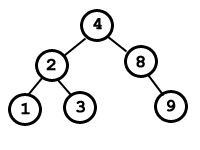


fig. 4.3

Se quiere insertar el elemento 6.

Lo primero es comparar el nuevo elemento con la raíz. Como 6 > 4, entonces la búsqueda prosigue por el lado derecho. Ahora el nuevo nodo se compara con el elemento 8. En este caso 6 < 8, por lo que hay que continuar la búsqueda por la rama izquierda. Como la rama izquierda de 8 no tiene ningún nodo, se cumple la condición de parada de la recursividad y se inserta en ese lugar el nuevo nodo.

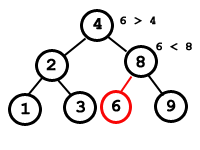


Fig. 4.3a

**Borrar:** El borrado en árboles binarios de búsqueda es otra operación bastante sencilla excepto en un caso. Vamos a ir estudiando los distintos casos.

Tras realizar la búsqueda del nodo a eliminar observamos que el nodo no tiene hijos. Este es el caso más sencillo, únicamente habrá que borrar el elemento y ya habremos concluido la operación.

Si tras realizar la búsqueda nos encontramos con que tiene un sólo hijo. Este caso también es sencillo, para borrar el nodo deseado, hacemos una especie de *puente*, el padre del nodo a borrar pasa a apuntar al hijo del nodo borrado.

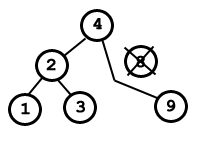


fig. 4.3b

Por último, el caso más complejo, si el nodo a borrar tiene dos hijos. En este caso se debe sustituir el nodo a borrar por mayor de los nodos menores del nodo borrado, o por el menor de los nodos mayores de dicho nodo. Una vez realizada esta sustitución se borra el nodo que sustituyó al nodo eliminado (operación sencilla ya que este nodo tendrá un hijo a lo sumo).

Sobre el siguiente árbol queremos eliminar el elemento 6. Tenemos dos opciones para sustituirlo:

* El menor de sus mayores: 7
* El mayor de sus menores: 4

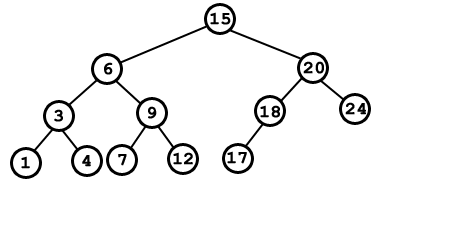


fig. 4.3c

Vamos a sustituirlo por el 7 (por ejemplo). El árbol resultante sería el siguiente, tras eliminar también el elemento 7 de su ubicación original.

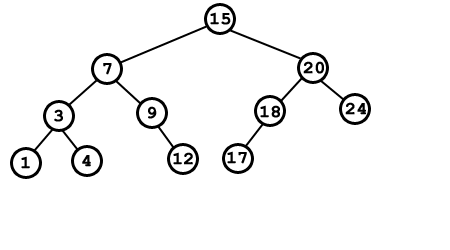


fig. 4.3d

**Otras operaciones**

En los árboles de búsqueda la operación buscar es muy eficiente. El algoritmo compara el elemento a buscar con la raíz, si es menor continua la búsqueda por la rama izquierda, si es mayor continua por la izquierda. Este procedimiento se realiza recursivamente hasta que se encuentra el nodo o hasta que se llega al final del árbol.

Otra operación importante en el árbol es el recorrido el mismo. El recorrido se puede realizar de tres formas diferentes:

* Pre orden: Primero el nodo raíz, luego el subárbol izquierdo y a continuación el subárbol derecho.
* In orden: Primero el subárbol izquierdo, luego la raíz y a continuación el subárbol derecho.
* Post orden: Primero el subárbol izquierdo, luego el subárbol derecho y a continuación la raíz.

4.4 APLICACIONES DE ÁRBOLES

Un árbol binario es una estructura de datos útil cuando se trata de hacer modelos de procesos en donde se requiere tomar decisiones en uno de dos sentidos en cada parte del proceso. Por ejemplo, supongamos que tenemos un arreglo en donde queremos encontrar todos los duplicados. Esta situación es bastante útil en el manejo de las bases de datos, para evitar un problema que se llama redundancia.

Una manera de encontrar los elementos duplicados en un arreglo es recorrer todo el arreglo y comparar con cada uno de los elementos del arreglo. Esto implica que si el arreglo tiene N elementos, se deben hacer N comparaciones, claro, no es mucho problema si N es un número pequeño, pero el problema se va complicando más a medida que N aumenta.

Si usamos un árbol binario, el número de comparaciones se reduce bastante, veamos cómo.

El primer número del arreglo se coloca en la raíz del árbol (como en este ejemplo siempre vamos a trabajar con árboles binarios, simplemente diremos árbol, para referirnos a un árbol binario) con sus subárboles izquierdo y derecho vacíos. Luego, cada elemento del arreglo se compara son la información del nodo raíz y se crean los nuevos hijos con el siguiente criterio:

* Si el elemento del arreglo es igual que la información del nodo raíz, entonces notificar duplicidad.
* Si el elemento del arreglo es menor que la información del nodo raíz, entonces se crea un hijo izquierdo.
* Si el elemento del arreglo es mayor que la información del nodo raíz, entonces se crea un hijo derecho.

Una vez que ya está creado el árbol, se pueden buscar los elementos repetidos. Si x el elemento buscado, se debe recorrer el árbol del siguiente modo: **Ejemplo 1:**

Sea k la información del nodo actual p. Si x > k   entonces cambiar el nodo actual a right(p), en caso contrario, en caso de que x = k  informar una ocurrencia duplicada y en caso de que x ≥ k cambiar el nodo actual a left(p).

El siguiente algoritmo:

Leer número buscado >> n

tree=makeTree(n)

while(hay números en el arreglo){

leeSiguienteNumero >> k

p=q=tree;

while(k!=info(p)&&q!=NULL){

p=q

if(k<info(p))

q=left(p)

else

q=right(p)

}

if(k==info(p))

Despliega<<" el número es duplicado";

else

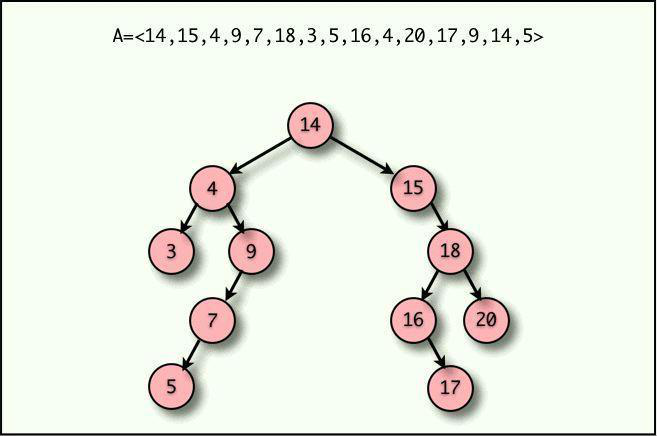
if (k<info(p))

setLeft(p,k)

else

setRight(p,k)

}



**Figura 28:** Árbol binario para encontrar números duplicados

Para saber el contenido de todos los nodos en un árbol es necesario **recorrer** el árbol. Esto es debido a que solo tenemos conocimiento del contenido de la dirección de un nodo a la vez. Al recorrer el árbol es necesario tener la dirección de cada nodo, no necesariamente todos al mismo tiempo, de hecho normalmente se tiene la dirección de uno o dos nodos a la vez; de manera que cuando se tiene la dirección de un nodo, se dice que sevisita ese nodo.

Aunque hay un orden preestablecido (la enumeración de los nodos) no siempre es bueno recorrer el árbol en ese orden, porque el manejo de los apuntadores se vuelve más complejo. En su lugar se han adoptado tres criterios principales para recorrer un árbol binario, sin que se omita cualquier otro criterio diferente.

Los tres criterios principales para recorrer un árbol binario y visitar todos sus nodos son, recorrer el árbol en:

**Pre orden:**

Se ejecutan las operaciones:

1. Visitar la raíz
2. recorrer el subárbol izquierdo en pre orden
3. recorrer el subárbol derecho en pre orden

**Entre orden:**

Se ejecutan las operaciones:

1. Recorrer el subárbol izquierdo en entre orden
2. Visitar la raíz
3. Recorrer el subárbol derecho en entre orden

**Post orden:**

Se ejecutan las operaciones:

1. recorrer el subárbol izquierdo en post orden
2. recorrer el subárbol derecho en post orden
3. Visitar la raíz

Al considerar el árbol binario que se muestra en la figura 28 usando cada uno de los tres criterios para recorrer el árbol se tienen las siguientes secuencias de nodos:

En pre orden: 

En entre orden: 

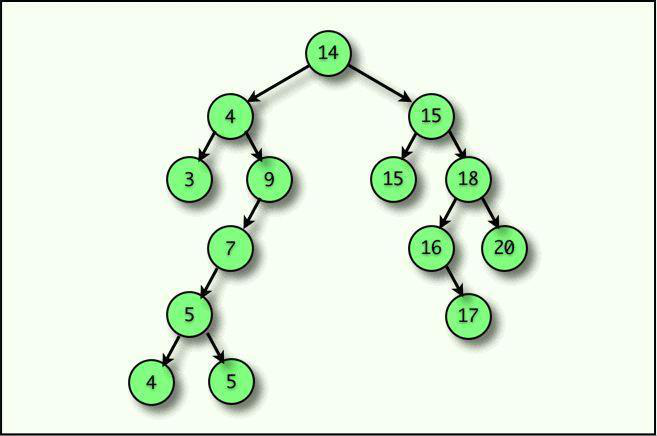
En post orden: 

Esto nos lleva a pensar en otra aplicación, el ordenamiento de los elementos de un arreglo.

Para ordenar los elementos de un arreglo en sentido ascendente, se debe construir un árbol similar al árbol binario de búsqueda, pero sin omitir las coincidencias.

El arreglo usado para crear el árbol binario de búsqueda fue: <14,15,4,9,7,18,3,5,16,4,20,17,9,14,5>

El árbol de ordenamiento es el que se muestra en la figura 29



**Figura 29:** Árbol binario para ordenar una secuencia de números

Para ordenar los elementos de este arreglo basta recorrer el árbol en forma de entre orden.

4.5 ÁRBOLES BALANCEADOS

**Equilibrio:**

La distancia de un nodo a la raíz determina la eficiencia con la que se puede localizar el nodo. Dado cualquier nodo de un árbol, a sus hijos se puede acceder siguiendo sólo un camino de bifurcación o de ramas. Esta característica se conoce como balance o equilibrio. Para determinar si un árbol está equilibrado, se calcula su factor de equilibrio.

El factor de equilibrio de un árbol binario es la diferencia en altura entre los subárboles derecho e izquierdo.

Un árbol está perfectamente equilibrado si su equilibrio o balance es cero y sus subárboles son también perfectamente equilibrados. Dado que esta situación no se presenta frecuentemente, se utiliza una definición alternativa que dice: un árbol binario está equilibrado si la altura de sus subárboles difiere en no más de uno y sus subárboles son también equilibrados.

Existe una definición de árbol perfectamente balanceado, que dice: "un árbol perfectamente balanceado es aquel en el que para cada nodo el número de nodos en sus subárboles derecho e izquierdo difieren como máximo en uno". Esta es la máxima expresión del equilibrio, sin embargo el costo (en procesamiento) para mantener este tipo de árboles es demasiado por ello la industria acepta y opera con el concepto de árbol equilibrado, en el que para cada nodo las alturas de los subárboles derecho e izquierdo difieren en no más de uno en valor absoluto.

Los siguientes esquemas muestran las diferencias entre cada concepto (definición):

Perfectamente Equilibrado:

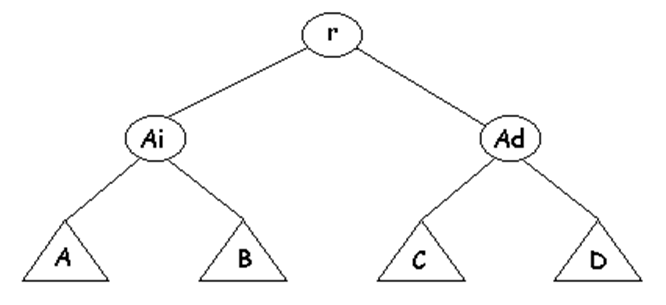


fig. 4.5

Perfectamente Balanceado:

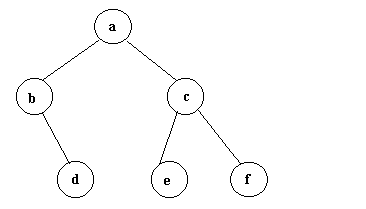


fig. 4.5a

Equilibrado:

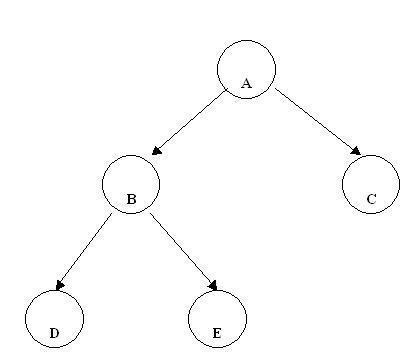


fig. 4.5b

Fueron Adelson-Velskii y Landis quienes propusieron una solución que se conoce como árbol AVL la que permite mantener un árbol equilibrado cada vez que se agrega o extrae un nodo. En esta solución cuando se realiza la búsqueda de un elemento, en el peor de los casos se requiere 44% más comparaciones que en un árbol perfectamente balanceado de igual altura, esto es así porque los subárboles difieren como máximo en 1 en sus alturas pero la cantidad de nodos puede ser muy variable. La industria prefiere esta implementación o sus variantes dado que es menos costosa a la hora de reorganizar el árbol de manera que mantenga la estructura adecuada.

4.6 GRAFOS

Un Grafo no es más que un conjunto de nodos o vértices que se encuentran relacionados con unas aristas. Además los vértices tienen un valor y en ocasiones las aristas también y se le conoce como el costo.

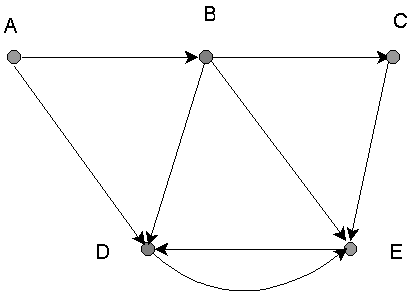


Fig. 4.6

Como se puede ver los puntos son los nodos o vértices, y las líneas son las aristas, en el caso de la imagen es la representación gráfica de un grafo dirigido ya que las aristas tienen un único sentido, ya que de A a D se puede ir pero de D a A no. Si el grafo fuera no dirigido se podría ya que las aristas no tienen dirección.

**Aplicación**

La teoría de Grafos se aplica hoy en día en muchos campos, tales como en Internet, ya que cada computador es un vértice y la conexión entre ellos son las aristas, además se usa para hallar la ruta más corta en empresas de transporte, y en muchas otras áreas.

4.7 TERMINOLOGÍA DE GRAFOS

Un grafo se compone por un conjunto de V vértices y un conjunto de A aristas. Cada arista se identifica con el par de vértices que une. Los vértices de una arista son entré si nodos adyacentes.

**Grado de un nodo:** Numero de aristas que contiene ese nodo. Si el grado de un nodo es 0, se dice que es un nodo aislado.

**Grado de un grafo:** Números de vértices de ese grafo.

**Camino:** Un camino C de longitud N de un nodo V1 a un nodo V2, se define como la secuencia de nodos por los que hay que pasar para llegar del nodo V1 a V2. La longitud es el número de aristas que comprende el camino. El camino es cerrado si empieza y termina en el mismo nodo. El camino es simple si todos los nodos de dicho camino son distintos a excepción de los de los extremos que pueden ser iguales.

**Bucles:** Aristas cuyos extremos son idénticos.

**Aristas múltiples:** Dos o más aristas que conectan los mismos nodos.

**Tipos de grafos**

**Grafo conectado o conexo:** Existe un camino simple entre dos cualquiera de sus nodos.

**Grafo desconectado:** Aquel en que existen nodos que no están unidos por ningún camino.

**Grafo dirigido:** Cada arista tiene asignada una dirección (identificada por un par ordenado).

**Grafo no dirigido:** La arista está definida por un par no ordenado.

**Grafo sencillo:** Aquel que no tiene ni bucles ni aristas múltiples.

**Grafo múltiple o multígrafo:** Permite la existencia de aristas múltiples o bucles.

**Grafo completo:** Cada nodo del grafo es adyacente a todos los demás.

**Grafo etiquetado con peso ponderado:** Cada arista tiene asociado un valor denominado peso. Se usa para indicar algún criterio de evaluación como la longitud o la importancia de la arista respecto a un parámetro.

**Peso de un camino:** La suma de los pesos de las aristas del camino.

4.8 OPERACIONES BÁSICAS SOBRE GRAFOS

En los grafos, como en todas las estructuras de datos, las dos operaciones básicas son insertar y borrar. En este caso, cada una de ellas se desdobla en dos, para insertar/eliminar vértices e insertar/eliminar aristas.

**Búsqueda de un nodo**

Se trata de localizar unos nodos que contiene una determinada información. Para ello le pasamos la información al procedimiento de búsqueda y este devuelve su posición si lo encuentra, o NULL en caso contrario.

**Búsqueda de una arista**

Se trata de buscar una arista dados sus campos de información origen y destino. Devuelve un puntero a la lista de aristas que apunta a esa arista o NULL si no existe.

**Operaciones básicas de los grafos**

En los grafos, como en todas las estructuras de datos, las dos operaciones básicas son insertar y borrar. En este caso, cada una de ellas se desdobla en dos, para insertar/eliminar vértices e insertar/eliminar aristas.

**Insertar vértice**

La operación de inserción de un nuevo vértice es una operación muy sencilla, únicamente consiste en añadir una nueva entrada en la tabla de vértices (estructura de datos que almacena los vértices) para el nuevo nodo. A partir de ese momento el grafo tendrá un vértice más, inicialmente aislado, ya que ninguna arista llegará a él.

**Insertar arista**

Esta operación es también muy sencilla. Cuando se inserte una nueva arista en el grafo, habrá que añadir un nuevo nodo a la lista de adyacencia (lista que almacena los nodos a los que un vértice puede acceder mediante una arista) del nodo origen, así si se añade la arista (A, C), se deberá incluir en la lista de adyacencia de A el vértice C como nuevo destino.

**Eliminar vértice**

Esta operación es inversa a la inserción de vértice. En este caso el procedimiento a realizar es la eliminación de la tabla de vértices del vértice en sí. A continuación habrá que eliminar las aristas que tuviesen al vértice borrado como origen o destino.

**Eliminar arista**

Mediante esta operación se borra un arco del grafo. Para llevar a cabo esta acción es necesario eliminar de la lista de adyacencia del nodo origen el nodo correspondiente al nodo destino.

**Otras operaciones**

Las operaciones adicionales que puede incluir un grafo son muy variadas. Además de las clásicas de búsqueda de un elemento o recorrido del grafo, también podemos encontrarnos con ejecución de algoritmos que busquen caminos más cortos entre dos vértices, o recorridos del grafo que ejecuten alguna operación sobre todos los vértices visitados, por citar algunas operaciones de las más usuales.

CONCLUSIÓN

Podemos decir, que los árboles son otra forma de estructura de datos no lineales. Se les considera un tipo de grafo pero sin ciclos además adopta otros términos como raíz, rama, hoja, etc. En sí, los árboles son estructuras que implican la relación de sus elementos a través de ramas, de ahí su nombre. Los arboles facilitan la búsqueda y ordenamiento de los datos y Su importancia radica en que los árboles son grafos que conectan todos los vértices utilizando el menor número posible de aristas.

Un importante campo de esta aplicación es el análisis filogenético, que se aplica a la averiguación del parentesco entre especies, aunque se ha usado también por ejemplo, en el estudio del parentesco entre lenguas.

De los grafos se puede decir que son sumamente importantes tanto en nuestra vida cotidiana como en la computación, ya que nos permiten organizar tareas o datos siguiendo una secuenciación, también representan enlaces entre estos.

Un grafo es básicamente un objeto geométrico aunque sea un objeto combinatorio, es decir, un conjunto de puntos y un conjunto de líneas. Podemos decir que a diferencia de los árboles en estos se incluyen ciclos además que no ocupan la misma terminología cuando hablados de sus partes.

Otro punto importante es que los grafos pueden representarse con una matriz de adyacencia y se le denomina, la forma más sencilla de representarlos y esto a su vez facilita las operaciones que se realizan en el grafo y más para quien este familiarizado con algebra lineal.

REFERENCIAS

<http://estructura-de-datos-itsav.blogspot.mx/2012/03/511-clasificacion-de-arboles.html>

<http://www.hci.uniovi.es/Products/DSTool/busqueda/busqueda-operaciones.html>

<http://www.algoritmia.net/articles.php?id=17>

Osvaldo Cairó, Estructura de Datos Tercera Edición, McGraw Hill, 2011