UNIREMINGTON®
CORPORACIÓN UNIVERSITARIA REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996

ESTRUCTURAS DE DATOS

INGENIERIA DE SISTEMAS

FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA

Vicerrectoría de Educación a Distancia y virtual

2016

UNIREMINGTON
CORPORACIÓN UNIVERSITARIA REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996

AUTOR

LUIS FERNANDO ZAPATA ALVAREZ
ESPECIALISTA EN GERENCIA INFORMATICA Y EN ADMINISTRACION DE LA INFORMATICA EDUCATIVA y MAGISTER EN GESTION DE LA TECNOLOGIA EDUCATIVA. Laboro en: Corporación universitaria Remington y docente de SEDUCA en la institución educativa Atanasio Girardot del Municipio de Girardota. MATERIAS QUE SIRVO en Un Unirémington en Ingeniería de sistemas: Introducción al desarrollo de software, Estructuras de datos, Sistemas Operativos y compiladores.
luis.zapata@uniremington.edu.co

Nota: el autor certificó (de manera verbal o escrita) No haber incurrido en fraude científico, plagio o vicios de autoría; en caso contrario eximió de toda responsabilidad a la Corporación Universitaria Remington, y se declaró como el único responsable.

RESPONSABLES

Jorge Mauricio Sepúlveda Castaño
Decano de la Facultad de Ciencias Básicas e Ingeniería
jsepulveda@uniremington.edu.co

Eduardo Alfredo Castillo Builes
Vicerrector modalidad distancia y virtual
ecastillo@uniremington.edu.co

Francisco Javier Álvarez Gómez
Coordinador CUR-Virtual
falvarez@uniremington.edu.co

GRUPO DE APOYO

Personal de la Unidad CUR-Virtual
EDICIÓN Y MONTAJE

Primera versión. Febrero de 2011.
Segunda versión. Marzo de 2012
Tercera versión. noviembre de 2015

TABLA DE CONTENIDO

Pág.

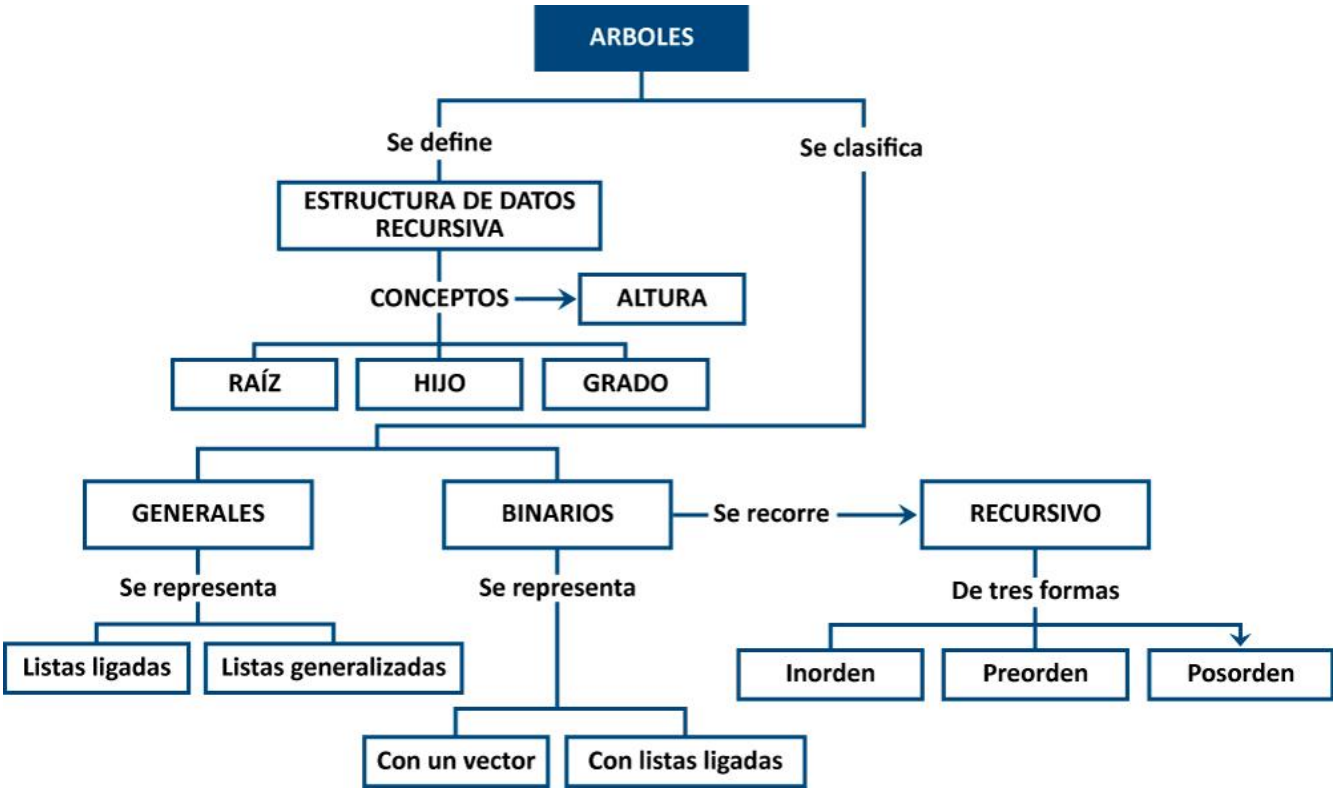
1 MAPA DE LA ASIGNATURA

2 UNIDAD 1 ARBOLES

La estructura árbol que es recursiva por definición es utilizada en diversos tipos de soluciones como por ejemplo la estructura de almacenamiento en disco de los archivos y directorios a partir el espacio particionado de disco en un sistema operativo.

UTPL ÁRBOLES [(INFORMÁTICA)(ESTRUCTURA DE DATOS)] Enlace

2.1.1 RELACIÓN DE CONCEPTOS



Raíz del árbol: Es principal concepto para la creación de la estructura, es el registro inicial con el que se crea el árbol.

Árbol general: Estructura recursiva con n>0 registros, en la cual una raíz puede tener 0, 1 , 2 o más registros que se derivan de ella.

Árbol binario: Estructura recursiva con n>=0 registros, en la cual una raíz pude tener 0, 1, 2 registros que se derivan de ella.

Registro padre: Es el registro que es raíz del árbol o de un subárbol de la estructura y tiene hijos.

Registro hijo: Es el registro que se deriva de otro registro del árbol

Estructura recursiva: la estructura se define en función de si misma

Grado: es el número de registros que se derivan del árbol

Nivel: el primer nivel es la raíz y si hay subárboles se va aumentando el nivel de a 1

Altura del árbol: Es el mayor nivel que alcanza el árbol

Inorden: recorrido del árbol binario que va primero al hijo izquierdo después la raíz y por último el hijo derecho

Preorden: recorrido del árbol binario que va primero a la raiz, después al hijo izquierdo y por último al hijo derecho.

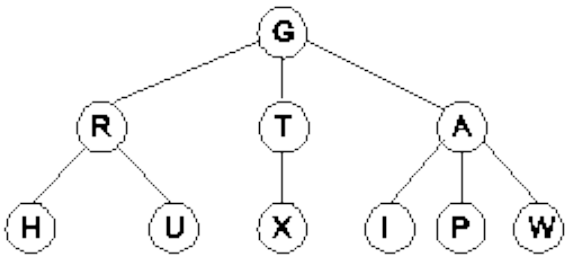
Posorden: recorrido del árbol binario que va primero al hijo izquierdo, después al derecho y por ultimo a la raíz.

2.2 TEMA 1 ARBOLES GENERALES Y SU REPRESENTACIÓN

La definición de árboles parte del concepto de árbol general que no incluye el árbol sin ningún registro. Además, se definen los conceptos básicos asociados a los árboles.

2.2.1 DEFINICIÓN DE ARBOLES GENERALES

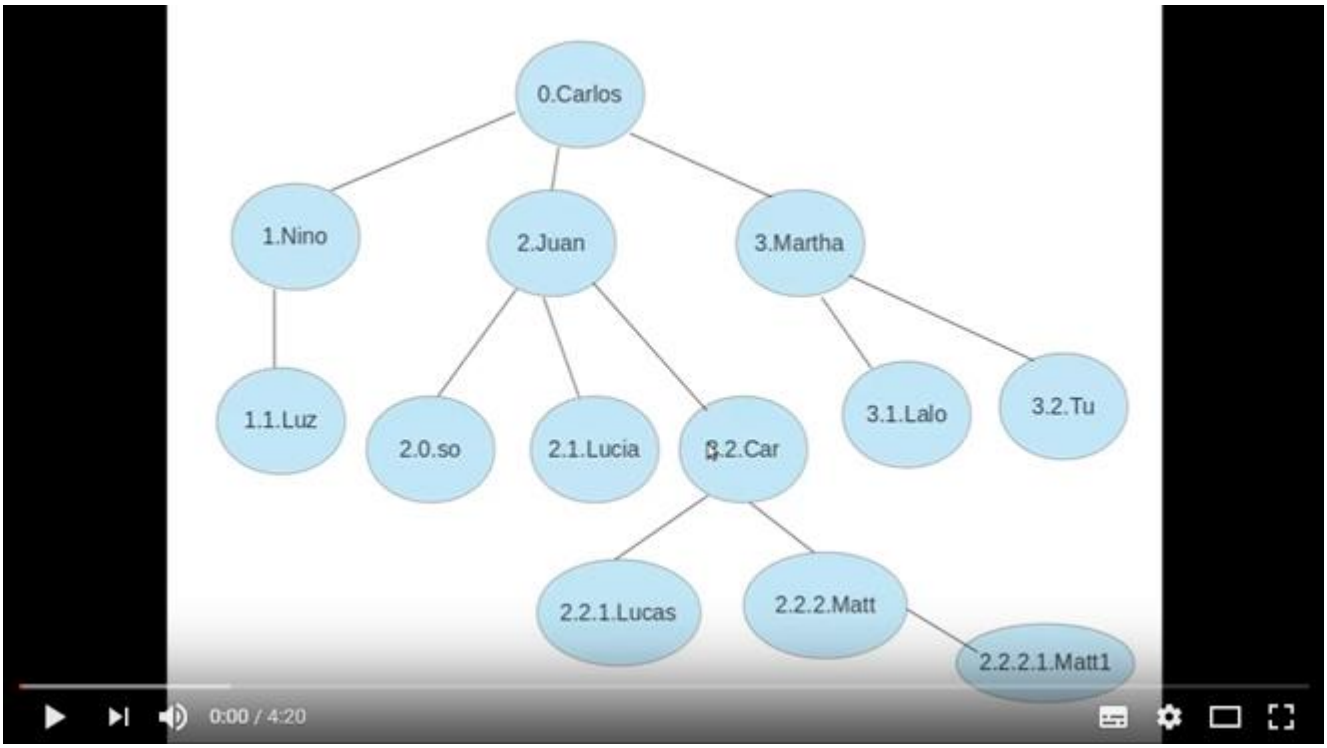
Un **árbol** es **un conjunto de n registros(n>0),** donde el **árbol vacío no está definido**, de tal manera que hay un registro llamado **raíz** y los otros registros están **partidos** en **conjuntos disjuntos** cada uno de los cuales tiene **las mismas características de la definición del árbol** (esta característica hace **comportar** la estructura árbol como **recursiva**). Gráficamente:



La **raíz principal** del árbol es **G** de ella se desprenden **tres árboles** recursivamente cuyas **raíces** son respectivamente **R, T, A** y así sucesivamente se pueden generar más árboles dependiendo de la estructura.

2.2.2 TERMINOLOGÍA DE ÁRBOLES

- Las ramificaciones de cada nodo se les denomina hijos y los nodos desde donde parten las ramificaciones se denominan padres. Los registros con el mismo padre se denominan hermanos.
- Al **número de ramificaciones** que tiene un registro se le denomina **el grado de un registro.**
- El **grado de un árbol** se determina encontrando **el registro** con el **más grande número** de **ramificaciones**.
- **Los registros** que **no tienen hijos** dentro del árbol se denominan **hojas**. Para calcular el número de hojas de un árbol ver el video en YouTube:



Calcular las Hojas en un Árbol N-ario Enlace

- El árbol tiene **niveles** que comienzan con el registro **raíz** y se denomina **nivel 1** y va aumentando de a uno con la ramificación de sus hijos.
- Un registro de **nivel k** tiene **el padre** en el **nivel k-1** e **hijos** en el **nivel k+1**
- Al **máximo nivel** alcanzado por el árbol se le denomina **la altura del árbol**.
- Para determinar **los ancestros** de **un registro** basta con encontrar **la trayectoria** desde **la raíz** al **registro** en cuestión.
- Otro nombre dado a este tipo de árboles, es **árbol n-arios**

2.2.3 REPRESENTACIÓN DE ARBOLES N-ARIOS

Se representan de dos formas: como listas ligadas y como listas generalizadas

2.2.3.1 REPRESENTACIÓN CON LISTAS LIGADAS

Para representar con listas ligadas se debe definir el registro teniendo en cuenta el grado del árbol.
El nodo debe tener una cantidad de apuntadores o de ligas igual al grado del árbol además de los

datos que almacene la estructura. Esto hace que la representación desperdicie memoria cuando un registro sea de grado inferior al del árbol y que además sea muy difícil de representar con listas pues cada árbol con grado distinto tiene una definición del nodo completamente distinta. Si el grado del árbol es 3 entonces el registro para la representación con listas tendría 3 campos de liga seria en ese caso:

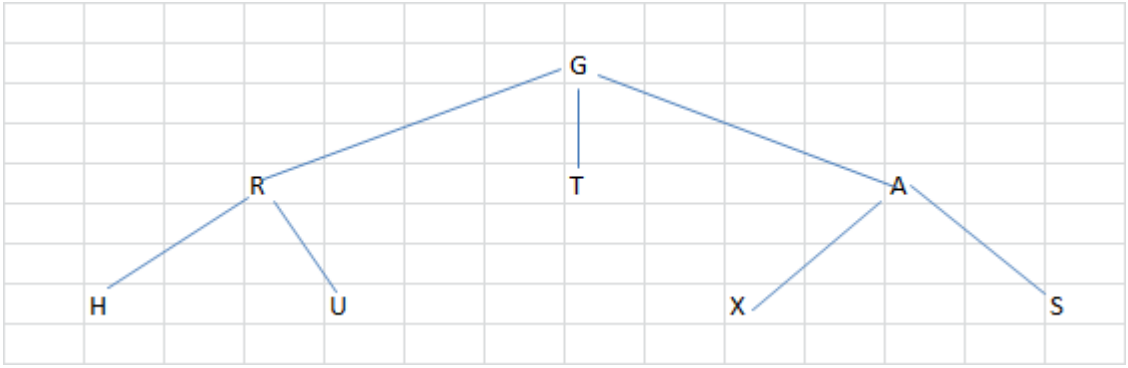
LIGA1	LIGA2	LIGA3	DATO

Ejemplo:

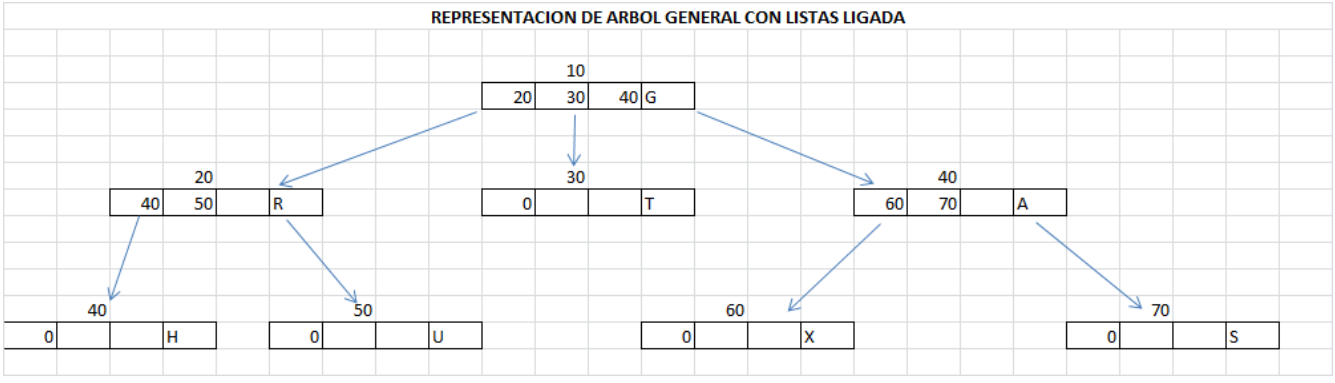
La representación del árbol para el ejemplo de arriba con listas ligadas será:

REPRESENTACION DE ARBOLES GENERALES CON LISTAS LIGADAS

Árbol:



Representación:



2.2.3.2 REPRESENTACIÓN CON LISTAS GENERALIZADAS

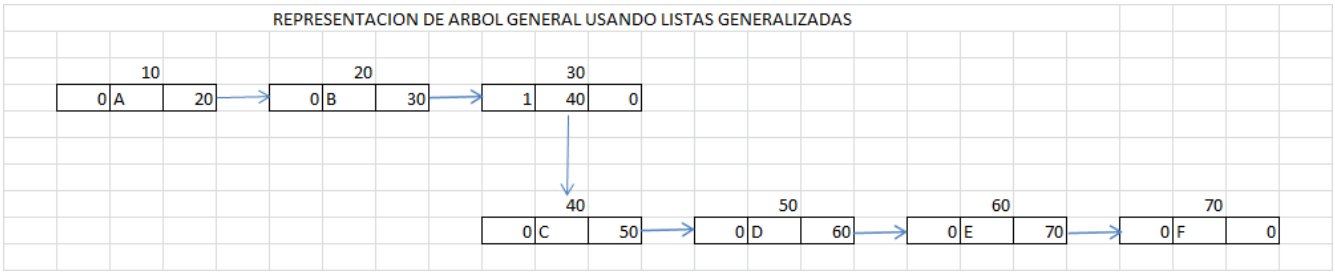
Para evitar los problemas de la representación con listas ligadas para los arboles generales recurrimos a las listas generalizadas cuya definición del nodo en la representación es:

Dónde: Si el sw =0 En el campo de dato hay un dato

Si sw=1 en el campo hay un apuntador hacia un sub árbol

La representación utiliza una lista simplemente ligada para cada raíz con sus hijos.

Ejemplo de representación de árbol general con listas generalizadas sea el árbol:



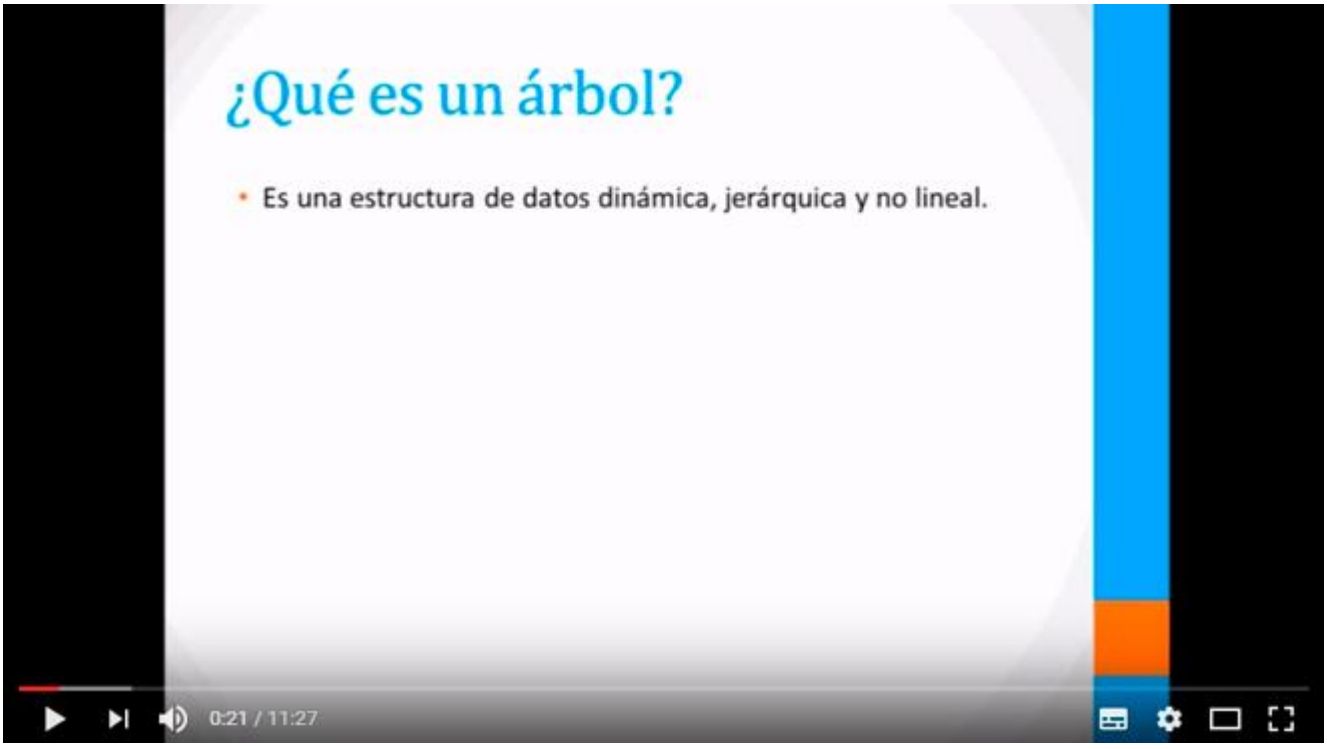
Explicación: La raíz principal forma una lista simplemente ligada con sus dos hijos (tres nodos), el concepto de lista generalizada aparece par el segundo nodo de la raíz principal que tiene el campo de sw=1 indicando que contiene en vez de un datos un apuntador a la segunda sublista que representa los cuatro nodos, la raíz y los tres hijos del subárbol de la derecha (en este todos tienen sw=0 y no hay mas sublistas)

2.3 TEMA 2 ARBOLES BINARIOS Y SU REPRESENTACIÓN

Este tipo de estructuras de datos son los fundamentales en la teoría de árboles, pues permiten realizar recorridos y búsquedas con la suficiente rapidez para lograr una más alta eficiencia en los algoritmos con los que se pueden resolver diversos procesos. Hay un tipo especial de árboles binarios que son los arboles de búsqueda que son creados como estructuras específicamente hechas para realizar búsquedas rápidas de información sobre la estructura de datos.

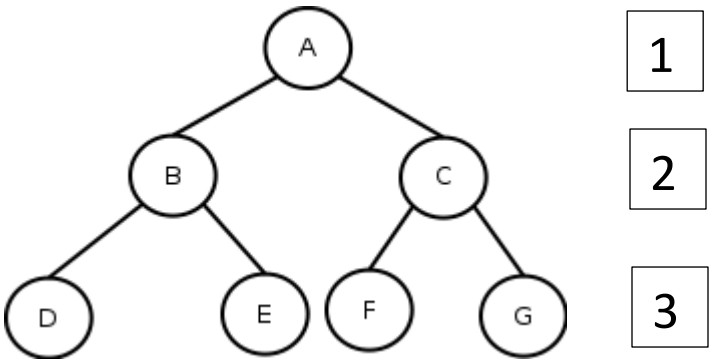
2.3.1 DEFINICIÓN DE ARBOLES BINARIOS

Un árbol binario es un conjunto de n registros (n>=0), el cual tiene una raíz o puede ser una estructura sin datos y en la cual los otros registros están partidos en dos conjuntos disjuntos llamados hijo izquierdo e hijo derecho que no son otra cosa que dos árboles también binarios. Es importante anotar que los arboles binarios son también una estructura de datos recursiva por definición, esto quiere decir que en su creación se debe asumir por donde se quiere crear el árbol y se debe llamar al procedimiento de inserción con su hijo izquierdo o derecho de acuerdo a hacia donde se esté creando el nuevo dato. Un ejemplo de árbol binario es:



Árboles Binarios de Búsqueda Enlace

09-Árboles de búsqueda binarios-02-Definición Enlace



La terminología usada para los arboles generales se usa de la misma forma con los arboles binarios, pero teniendo en cuenta que el grado del árbol binario es 2. Por ejemplo:

El registro D es hijo del registro B y nieto del registro A, mientras que D, E, F y G son hermanos, el grado del árbol binario es 2 y la altura del árbol es 3 pues tiene tres niveles.

2.3.2 PROPIEDADES DE LOS ARBOLES BINARIOS

Se pueden presentar las siguientes propiedades de este tipo de árboles:

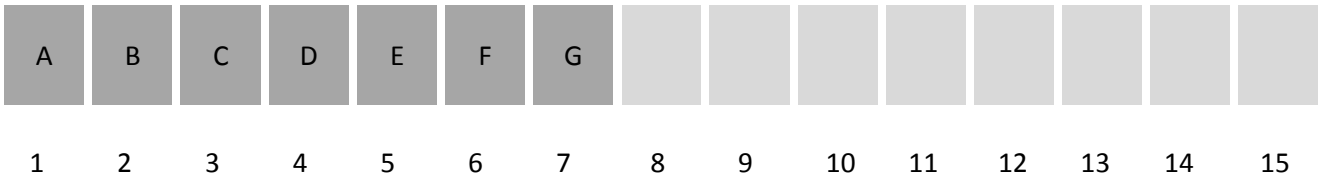
- El número máximo de registros de un nivel k del árbol es 2k-1
- Un árbol binario lleno: es un árbol de altura k que tiene 2k -1 registros, ósea cada registro tiene sus dos hijos hasta el último nivel que solo hay hojas
- Sea n0 = al número de hojas del árbol y n2 número de registros de grado 2. Siempre se cumplirá que n0=n2+1, pero no el árbol vacío.

2.3.3 REPRESENTACIÓN DE LOS ARBOLES BINARIOS

Se pueden representar como un vector de forma estática o como una lista ligada de forma dinámica. Veamos cada una de estas dos representaciones:

2.3.3.1 REPRESENTACIÓN DE ÁRBOLES BINARIOS CON UN VECTOR

El nivel 1 corresponde a una posición donde colocamos el dato de la raíz principal del árbol, a las dos posiciones siguientes les corresponde los datos de los dos registros hijos del nivel 2, las cuatro siguientes posiciones del vector serán para los dos pares de hijos del nivel 3, de tal manera que el siguiente nivel tendrá 8 posiciones del vector para representar la cantidad posible de hijos del 4 nivel del árbol y así sucesivamente como una potencia de 2. Gráficamente para el árbol del ejemplo inicial:



Por tener el árbol solo tres niveles el tamaño de la estructura estática será de 7 un dato en el primer nivel, dos datos en el segundo nivel y cuatro datos en el tercer nivel del árbol. En el caso de que el árbol no tenga alguno de los hijos de los dos lados el árbol, no ubica en la posición correspondiente al nodo faltante en el vector ninguna información.

2.3.3.2 REPRESENTACIÓN DE ÁRBOLES COMO UNA LISTA LIGADA

Para representar la estructura árbol binario de manera dinámica se requiere definir el registro del nodo de la siguiente manera:

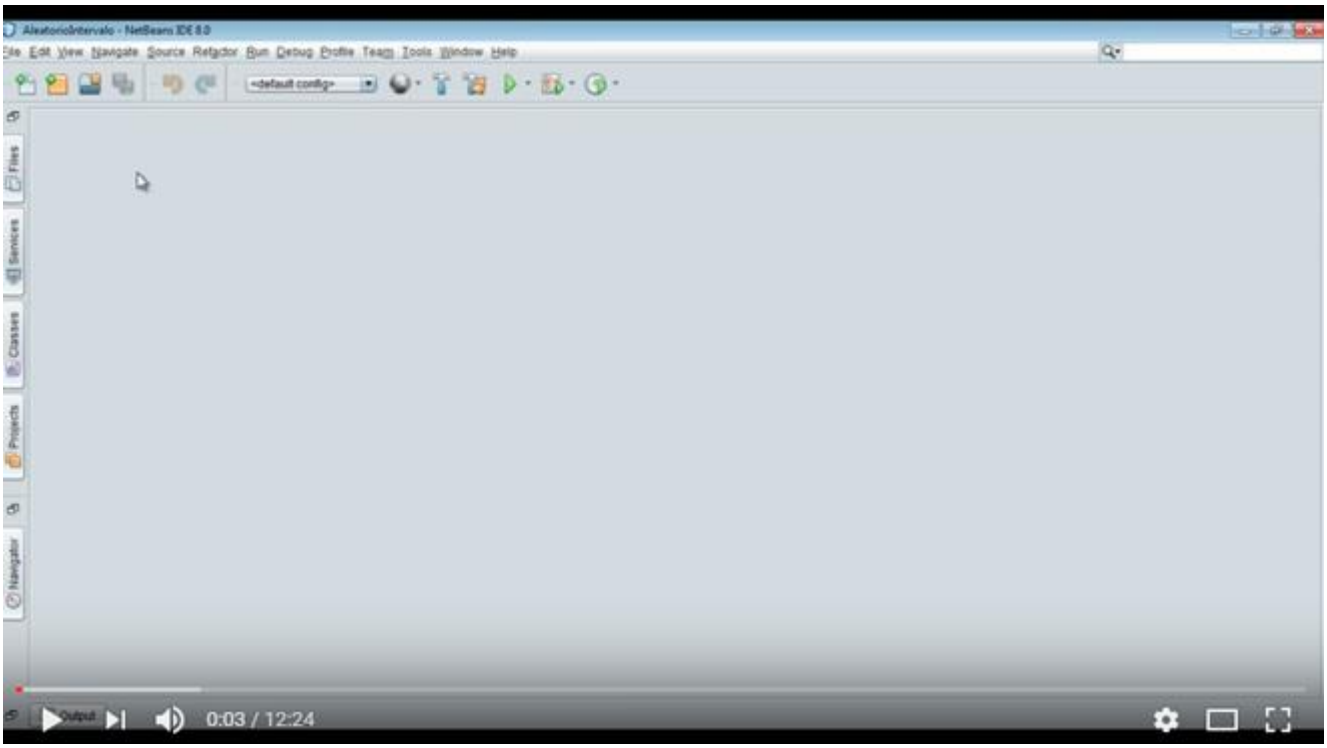
Con:

LI: Apuntador al subárbol izquierdo del árbol

LD: Apuntador al subárbol derecho del árbol

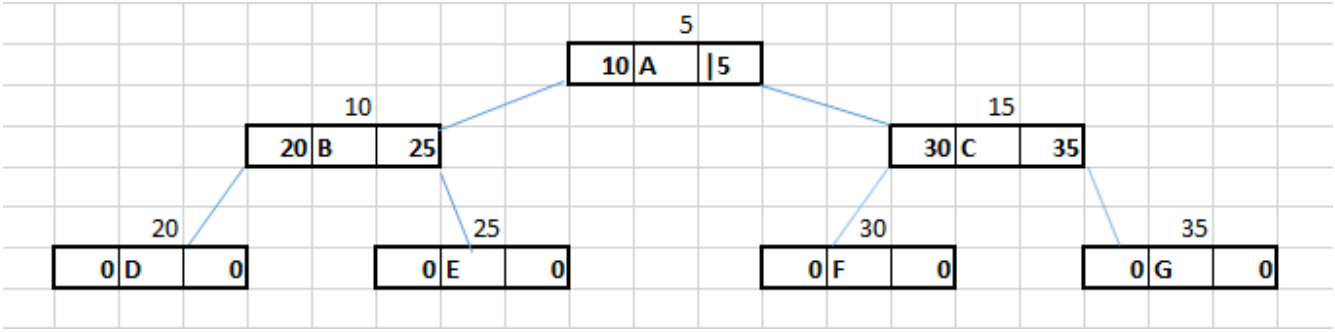
D: Representa el dato almacenado en la estructura (este puede ser también un registro si es necesario almacenar más datos).

34 - Árboles Binarios de Búsqueda, Creación e Inserción (EDDJava) Enlace



Árbol Binario de Búsqueda Implementado en Java Enlace

Para el árbol del ejemplo la representación quedaría:



Las hojas siempre deben tener las dos ligas iguales a 0 (Esto se debe tener en cuenta para el algoritmo cuenta hojas en arboles binarios).

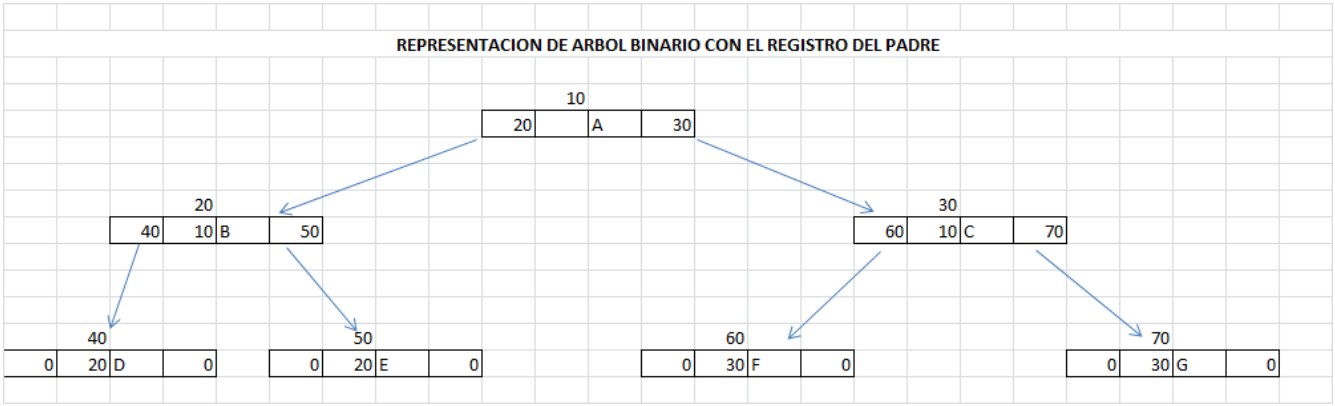
2.3.3.3 REPRESENTACIÓN DE ÁRBOLES COMO UNA LISTA LIGADA CON LA DIRECCIÓN DEL PADRE

Esta representación difiere de la anterior por que la definición del nodo tiene un campo más para el registro que es un apuntador con la dirección del padre. Así:

LI	RP	D	LD

LI, LD, D: Son la misma definición anterior y RP: Es la dirección del registro del padre para cada nodo representado. Si el registro es la raíz del árbol no tendrá RP. Esta representación es muy útil cuando se usan algoritmos donde con frecuencia debemos regresar al registro del padre de cualquiera de los nodos del árbol.

La representación del árbol binario anterior con el registro del padre es:

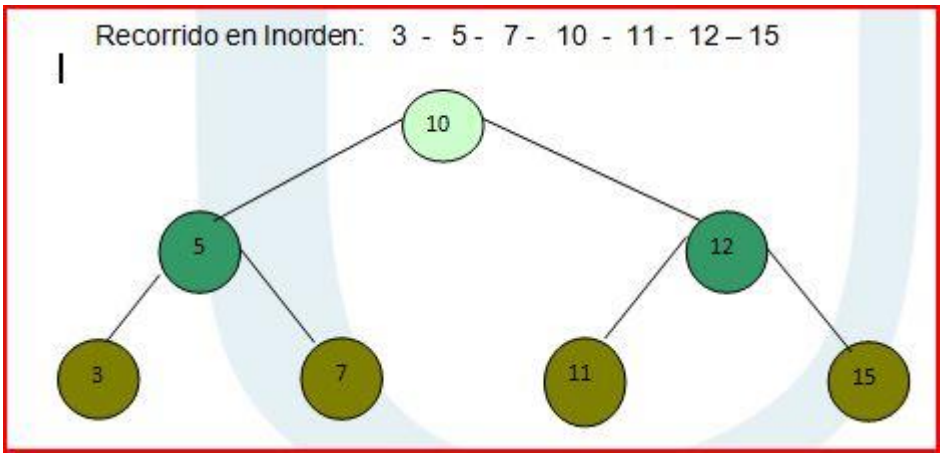


Explicación: La diferencia en la anterior representación es que cada registro exceptuando al padre tienen la dirección del registro padre como segundo campo del nodo representado.

2.3.3.4 RECORRIDOS SOBRE ARBOLES BINARIOS

Teniendo en cuenta que los recorridos sobre arboles binarios dan la posibilidad de moverse en la estructura de datos para realizar cualquier tipo de operación que permita su modificación o actualización. Se dan tres recorridos principales en los arboles binarios: (Se debe tener en cuenta que cuando se representan operaciones en un árbol binario la raíz siempre tiene al operador y los hijos son los operados)

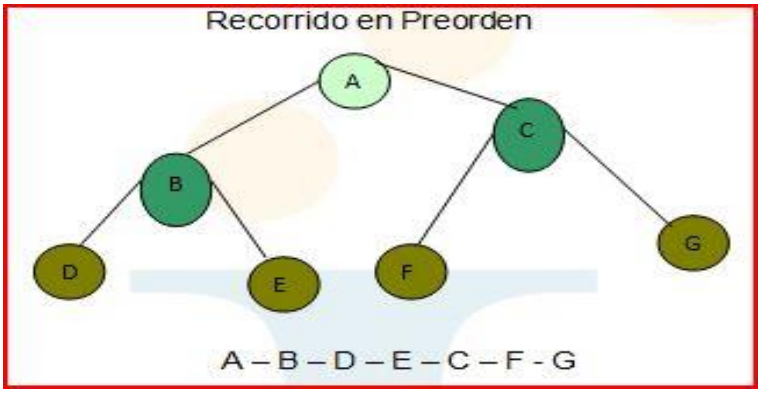
- Recorrido Inorden: Los recorridos comienzan por la dirección de la raíz principal del árbol binario y consiste en visitar primero el hijo izquierdo, segundo imprimir el dato de la raíz y por ultimo visitar el hijo derecho, su forma de representación es: IRD (IZQUIERDO, RAIZ, DERECHO), es de anotar que los recorridos son recursivos como son también los árboles, es por eso que el llamado a los hijos izquierdo y derecho del árbol es llamar a otro subárbol binario. Un ejemplo para este recorrido:



Explicación: partimos de la raíz principal que es 10, pero como el recorrido primero va a la izquierda pasamos al subárbol de raíz 5 pero a su vez este también tiene subárbol izquierdo que es el registro hoja 3, como este no tiene hijos se escribe la raíz que es 3 y se regresa a la raíz 5 que también se escribe pues ya se visitó su hijo izquierdo y se pasa a su hijo derecho que también es una hoja y se imprime la raíz. Como ya se visitó el hijo izquierdo se regresa a la raíz principal y se imprime 10 pasando a recorrer el hijo derecho del árbol que debe imprimir respectivamente 11, 12 y 15 con el mismo análisis anterior.

35 - Árboles Binarios de Búsqueda, Recorrido InOrden (EDDJava) Enlace

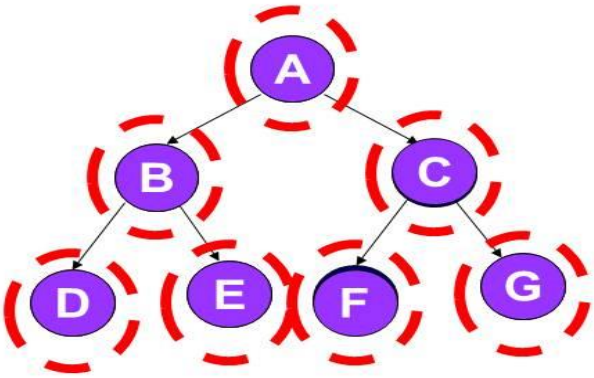
- Recorrido preorden: Consiste en imprimir el dato de la raíz y después visitar al hijo izquierdo y posteriormente al derecho como dos llamados recursivos a los subárboles respectivos, su forma de representación es: RID (RAIZ, IZQUIERDO, DERECHO). Un ejemplo para este recorrido es:



36 - Árboles Binarios de Búsqueda, Recorrido PreOrden (EDDJava) Enlace

- Recorrido posorden: Consiste en recorrer inicialmente los hijos izquierdo y derecho para dejar de ultimo la impresión del dato de la raíz, su forma de representación es: IDR (IZQUIERDO, DERECHO y RAIZ). Un ejemplo de este recorrido es:

Recorrido en Postorden

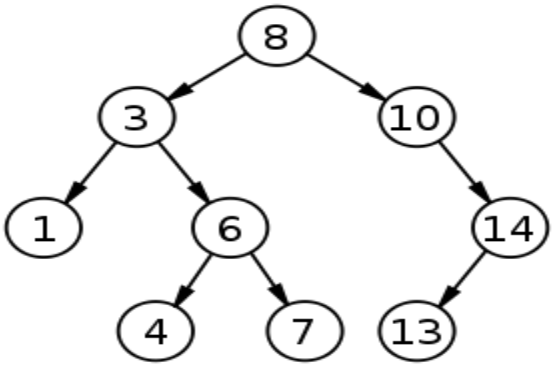


Recorrido D E B F G C A

1. Subárbol izquierdo en orden final.
2. Subárbol derecho en orden final.
3. Raíz.

37 - Árboles Binarios de Búsqueda, Recorrido PostOrden (EDDJava) Enlace

- Ejemplo de un árbol con los tres recorridos definidos:



Inorden: 1 **3 4 6 7 8 10 13 14**

Preorden: **8 3 1 6 4 7 10 14 13**

Posorden: 1 4 7 6 3 13 14 10 8

- Algoritmos para los recorridos de árboles binarios: Se definen los algoritmos para una representación de árboles con listas ligadas, donde la raíz es de tipo nodo Float R y se utiliza dentro de la clase árbol un método que puede devolver el hijo izquierdo y el hijo derecho o el dato del nodo para determinar el recorrido recursivo respectivo:

```
Inorden(nodoFloat R)
        If(R!=0) then
                Inorden(R.devli())          //llamado recursivo con LI(R)
                Imprima(R.devDato())        //imprime el dato(R)
                Inorden(R.devld())          //llamado recursivo con LD(R)
        End(if)
Fin(Inorden)

Preorden(nodoFloat R)
        If(R!=0) then
                Imprima(R.devDato())        //imprime el dato(R)
                Preorden(R.devli())         //llamado recursivo con LI(R)
                Preorden(R.devld())         //llamado recursivo con LD(R)
        End(if)
Fin(Preorden)

Posorden(nodoFloat R)
```

```
If(R!=0) then
        Posorden(R.devli())        //llamado recursivo con LI(R)
        Posorden(R.devld())        //llamado recursivo con LD(R)
        Imprima(R.devDato())       //imprime el dato(R)
End(if)
Fin(Posorden)
```

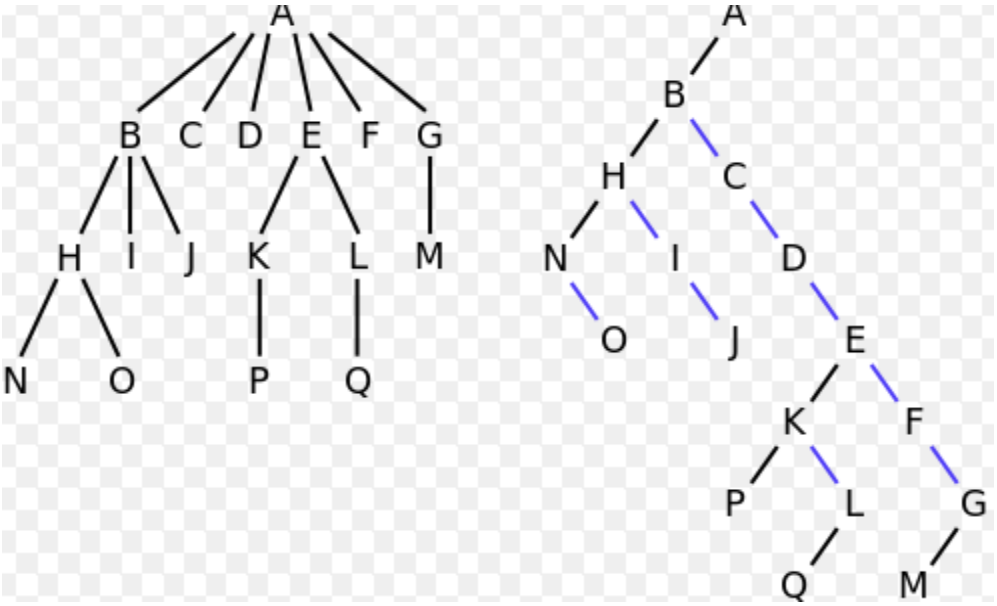
Nota: para realizar las pruebas de escritorio a estos algoritmos se debe manejar la estructura pila con las direcciones de retorno a los llamados respectivos en los métodos que se deben apilar cuando se hace el llamado y posteriormente se desapilan cuando termina el llamado.

Adicionalmente todos estos algoritmos se pueden escribir en un lenguaje de programación como java para simular la creación de esta estructura de datos como ya fue ilustrado en videos anteriores en este mismo capítulo.

2.3.3.5 REPRESENTACIÓN DE UN ÁRBOL GENERAL COMO UN ÁRBOL BINARIO

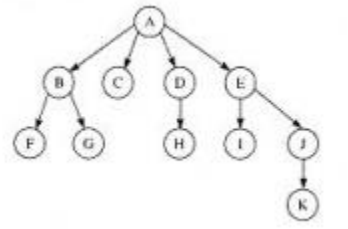
Cualquier árbol general sin importar su grado, puede ser representado como un árbol binario. La construcción del árbol binario es como sigue: Para cada registro r que tenga k hijos, su primer hijo se representa como hijo izquierdo y los otros se insertan como hijos derechos del primero respectivamente. Lo que quiere decir que los registros que están en una rama derecha del árbol son los hermanos de la raíz que los contiene. Ejemplo:

Para el siguiente árbol General **Esta es la Representación como un árbol binario**



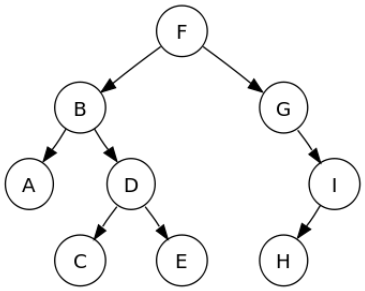
2.3.4 TALLER DEL CAPÍTULO:

1. Para el siguiente Árbol general



Representar como: Listas ligadas y listas generalizadas

2. Para el siguiente árbol binario:



Representar como listas ligadas y como listas ligadas con el registro del padre

3. Describir los recorridos inorden, postorden y preorden asociados al árbol del punto 2
4. Convertir el árbol del punto 1 en binario
5. Escribir un algoritmo que cree un árbol binario de forma recursiva
6. Hacer un seguimiento recursivo en Inorden para el árbol que tiene tres registros utilizando la pila
7. Escribir un algoritmo que busque un dato en que puede encontrarse dentro del árbol.

2.4 TEMA 3 LISTAS GENERALIZADAS

Es una estructura que permite escribir listas ligadas dentro de otras cuando la necesidad de manejo de información lo requiera. Un ejemplo importante son los polinomios de varias variables en la teoría matemática o una forma de representar árboles generales como listas generalizadas. Es posible que algún tipo de estructuras recursivas se puedan representar como una lista generalizada.

2.4.1 DEFINICIÓN DE LISTAS GENERALIZADAS

Es un conjunto finito de n elementos (n>=0) cada uno de los cuales representa un dato almacenado o apunta a otra lista generalizada, es una estructura que se define así misma ósea que es recursiva. Los datos que puede representar se denominan átomos. Realizando una representación de acuerdo a la teoría formal de conjuntos la notación usada será: letras mayúsculas representan listas y letras minúsculas representan átomos. Un ejemplo seria:

A={x,y}

C={a,A,c}

B={d,C,f,A}

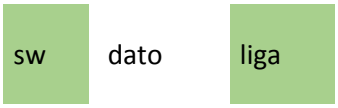
Se pueden mostrar los conjuntos C y B por extensión:

C={a,(x,y},c}

B={d,{a,A,c},f, {x,y}}= {d,{a,{x,y},c},d, {x,y}}

2.4.1.1 REPRESENTACIÓN DE LISTAS GENERALIZADAS

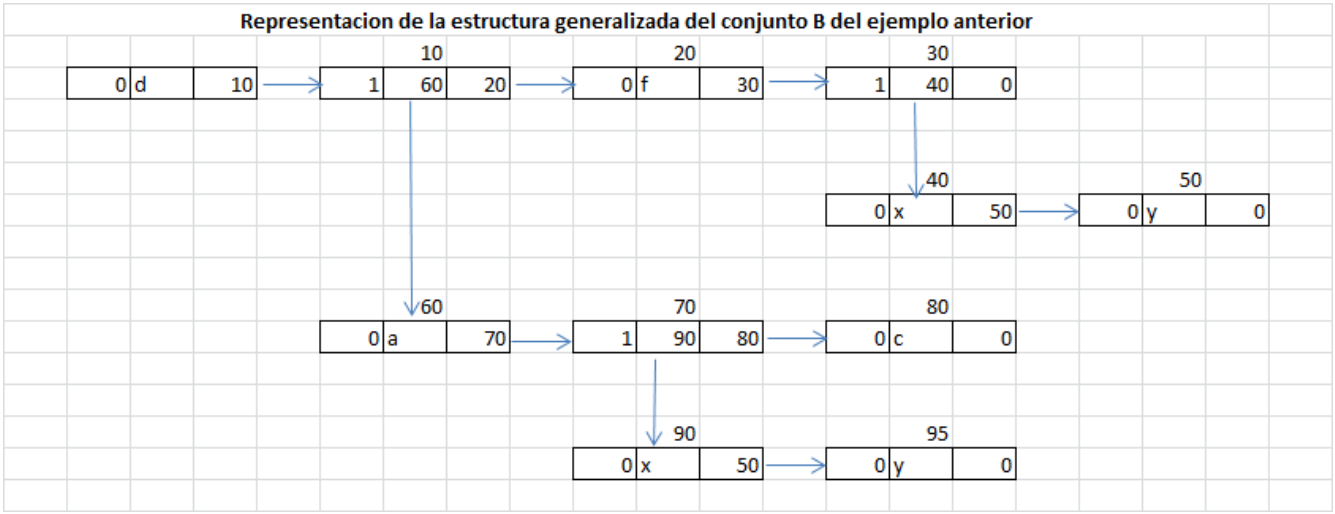
Su forma de representación básica es una lista ligada que define el registro de la lista como sigue:



El sw se define de la si8guiente forma:

- 0: si en el campo de dato hay un átomo
- 1: si en el campo de dato hay un apuntador a una sublista

Cada elemento de la lista generalizada utiliza un nodo. A continuación, se representa la lista del ejemplo propuesto en la representación de conjuntos:



2.4.1.2 CONSTRUCCIÓN DE UNA LISTA LIGADA QUE REPRESENTA UNA LISTA GENERALIZADA

Se debe tener en cuenta la forma en que se define la hilera de entrada que representa la lista generalizada así: paréntesis abierto, átomos, comas, paréntesis cerrado. El algoritmo recibe como parámetro de entrada la hilera h que representa el conjunto:

```
Void conslg(string s)
        Stack pila=new stack()    //estructura pila para las sublistas
        X=new nodolg(null)        //consigue nuevo nodo de la lista a crear
        L=x                       //inicialización de punteros de la lista
        Ultimo =x
        n=longitud(s)             // n contiene el tamaño de la cadena s
        for(i=2;i<n;i++) do       //ciclo para recorrer  la cadena
                Casos para s[i]                  //casos para determinar que viene en la cadena
                        Átomo:
                                Ultimo.asignasw(0)          //crea el átomo en la lista
                                Ultimo.asignadato(s[i])
                        “,”     x=new nodolg(null)          //crea otro nodo para lo que venga
                                Ultimo.asignaliga(x)        //en la lista a continuación
                                Ultimo=x
                        “(“     pila.apilar(ultimo)         //se debe construir una sublista
                                x=new nodolg(null)
                                Ultimo.asignasw(1)
                                Ultimo.asignadato(x)
                                Ultimo=x
                        “)”     ultimo=pila.desapilar       //se terminó de la construcción de la
                                                            //sublista y se debe regresar a la
```

//construcción que se interrumpió

Nota: una de las aplicaciones básicas de las listas generalizadas se da en la representación de polinomios con muchas variables. La definición del registro en este caso:

Sw	coef	exp	liga

Teniendo en cuenta que el sw se define como:

0: si el campo coef es un escalar

1: si el campo es un apuntador al polinomio que es coeficiente.

<Escriba aquí el contenido del subtema. Agregue tantos bloques de título y contenido como requiera.>

Ejercicios propuestos sobre listas generalizadas:

1. Representar con listas generalizadas el siguiente polinomio:

   $3x^2+(2y+3)x-2$

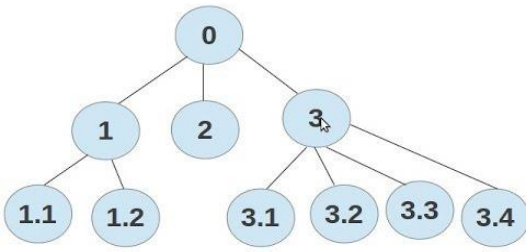
2. Representar como lista generalizada el siguiente conjunto al conjunto C:

   A=(b, B, d, e)

   B=(f,g)

   C=(z, A,B)

3. Representar con listas generalizadas el siguiente árbol n-ario

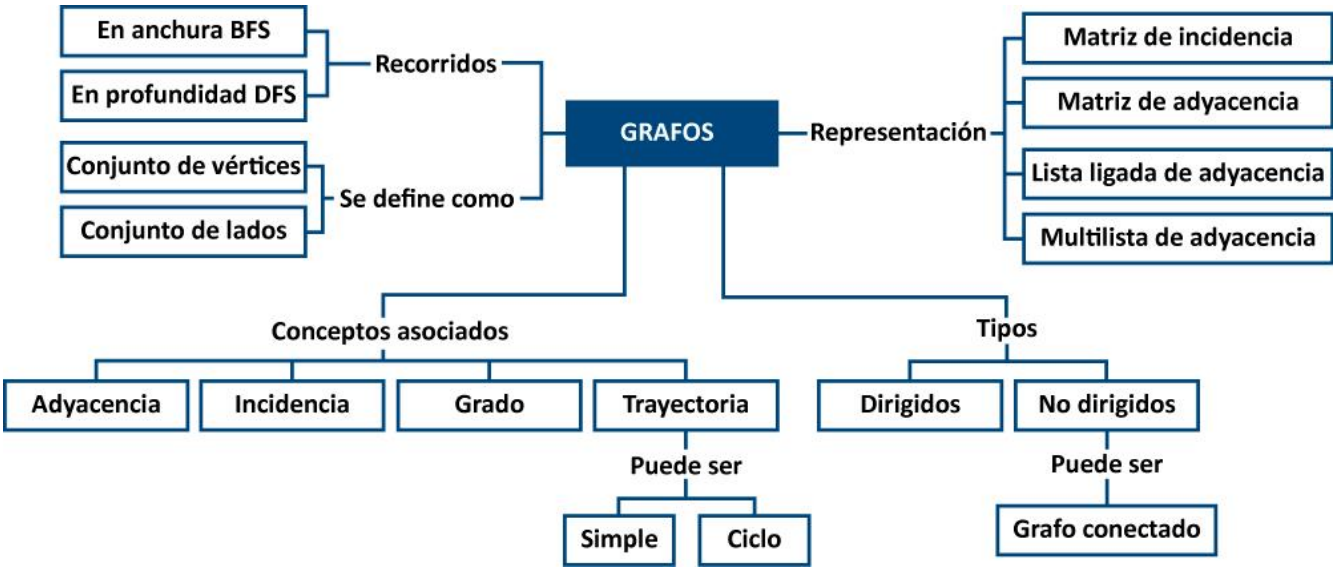


3 UNIDAD 2 GRAFOS

Los grafos son estructuras de datos que se utilizan en diversas aplicaciones, especialmente en problemas de transporte que requieren encontrar rutas más cortas entre ciudades o países para optimizar él envió de productos o que permita reducir los tiempos de envió entre dos puntos de distinta ubicación. Hay algún tipo de grafos específicos (no dirigidos) que son equivalentes a la estructura árbol. Como ejemplos de grafos se pueden mencionar: Las rutas de una empresa de transporte, una red de computadoras, el sistema vial de un país, los sitios de interés de una agencia de viajes, etc.

Teoría de Grafos en la vida real. Árboles. Árboles dirigidos con raíz.© UPV Enlace

3.1.1 RELACIÓN DE CONCEPTOS



3.1.2 DEFINICIÓN DE CONCEPTOS

Adyacencia: conformación de un lado

Conectado: Para un grafo no dirigido es la propiedad de ir a todos los otros vértices desde eses vértice

Grado de un vértice: cantidad de lados que salen del vértice

Grafo dirigido: El lado que relaciona dos vértices es una flecha dirigida, mostrando la relación

Grafo no dirigido: El lado es la doble relación entre los vértices y se conecta con una línea

Grafo: Conjunto de n vértices y m lados

Incidencia: El lado que forman dos vértices es incidente sobre ellos

Lado: es la unión de dos vértices

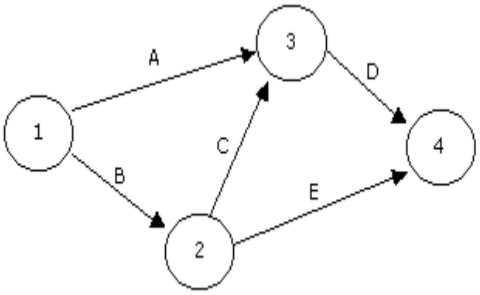
Trayectoria: Recorrido para ir de un vértice i a un vértice j

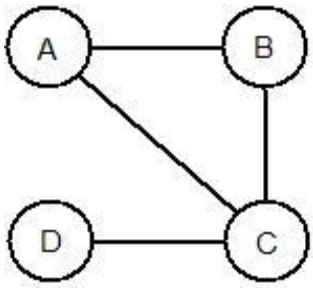
3.2 TEMA 1 DEFINICIÓN Y TERMINOLOGÍA BÁSICA SOBRE GRAFOS

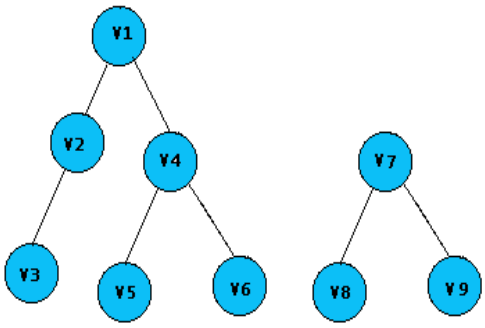
3.2.1 DEFINICIÓN DE GRAFOS

Se define como un **conjunto finito de puntos o vértices** que se comunican con **una traza** llamada **lado** para formar una figura con estas relaciones (un grafo contiene **un conjunto finito de lados**).

Ejemplos:







Grafo 1 **Grafo 2** **Grafo 3**

Los vértices y lados para los grafos 1 y 2 son:

Grafo 1: V1={1,2,3,4}

L1={<1,2>,<1,3>,<2,3>,<2,4>,<3,4>}

Grafo 2: V2={A,B,C,D}

L2={(A,B),(A,C),(B,C),(C,D)}

3.2.1.1 CLASIFICACIÓN

Los grafos se clasifican así:

GRAFOS	CARACTERÍSTICAS
Grafos no dirigidos	Se caracterizan por que **sus lados no están orientados**, se representan entre **paréntesis**. Como

	en los grafos 2 y 3 donde los vértices **(v1, v2)** es igual al vértice **(v2, v1)** que conforman **el mismo lado**.
Grafos Dirigidos	Son los que tienen **sus lados orientados**, gráficamente se realiza con **una flecha** indicando hacia dónde **va dirigido, el lado** y **los vértices** se representan entre **ángulos.** El grafo 1 es dirigido y el vértice **<1,2>** es **diferente** del vértice **<2,1>.**
Nota: Cuando se **trata de un grafo dirigido cada vértice de un lado se diferencia de la siguiente forma:** **<V_i,V_j>: V_i: cabeza del lado**; V_j: **cola del dado**	

3.2.1.2 TERMINOLOGÍA BÁSICA DE GRAFOS

- **Adyacencia**: Dos vértices son adyacentes si conforman un lado: por ejemplo en el grafo 1, los vértices 1 y 2 son adyacente, también los son 1 y 3.

La adyacencia en los grafos dirigidos se da de dos formas según la orientación del grafo:

<V_i,V_j>: V_i es adyacente hacia V_j y que V_j: Es adyacente desde V_i.

- El lado que forman dos vértices es incidente sobre ellos
- **Grado de un vértice**: Es el número de lados incidentes sobre él. En el Grafo 3, el vértice 3 tiene grado=1; el vértice 1 tiene grado=2.
- En el caso de los grafos dirigidos se define:

Grado entrante: Número de lados que llegan al vértice

Grado Saliente: Número de lados que salen del vértice

El **grado total** es **la suma** del **grado entrante** más el **grado saliente**

- **Trayectoria:** describe el camino para ir de un vértice i a un vértice j en un grafo. Por ejemplo en el grafo 1 para ir del vértice 1 al 4 puedo ir con tres trayectorias: **1234, 134, 124.** Es de anotar que para que exista la trayectoria los lados sobre la trayectoria deben pertenecer al conjunto de lados del grafo. Así <1,2>, <2,3>,<3,4> pertenecen al conjunto de lados del Grafo 1, lo anterior quiere decir que en un grafo pueden haber trayectorias que no son válidas.

- **Longitud de una trayectoria:** Se define con la cantidad de lados que contiene. En el ejemplo anterior para la trayectoria 1234 su longitud es: 3. En el caso de 134 y 124 la longitud en cada caso es: 2

- **Trayectoria Simple:** Se da cuando todos los vértices excepto posiblemente el primero y el ultimo son distintos. Por ejemplo: 1234 es trayectoria simple en el Grafo 1. Pero si se diera la trayectoria 12324 en este grafo no sería una trayectoria simple. Otro caso analizable seria la trayectoria 12341 que efectivamente seria simple (el primero y el último de los vértices son iguales).

- **Ciclo:** Se define como una trayectoria simple en el cual el primero y el último vértice son iguales. En el Grafo 2 la trayectoria ABCA forma un ciclo.

- **Grafo conectado**: se denomina así si desde cualquier vértice i del grafo se puede ir a cualquier vértice j del grafo (para grafos no dirigidos). Para grafos dirigidos se usa el concepto grafo fuertemente conectado con la misma definición que el anterior.

- **El máximo número de lados en un grafo no dirigido se calcula como**: **n*(n-1)/2** con n igual al número de vértices del grafo.

- **Un grafo dirigido completo tiene un número de lados igual a**: **n*(n-1)** con n igual al número de vértices

3.2.1.3 REPRESENTACIÓN DE GRAFOS:

Hay **4 formas** de representar los grafos entre las representaciones estáticas y dinámicas que se describen a continuación:

- **Matriz de adyacencia:** Es una matriz cuadrada de orden n*n, siendo n el número de vértices del grafo. La relación que se da entre los vértices del grafo se representa en la matriz teniendo en cuenta lo siguiente:

  Si existe lado (V_i, V_j) el cruce entre i y j se llena con 1 (hay adyacencia)

  En otro caso es 0 (se deja el cruce en blanco).

  A continuación se representa el Grafo 1 y el Grafo 2 como matriz de adyacencia:

Representación Grafo 1

	1	2	3	4
1		1	1	
2			1	1
3				1
4				

Representación Grafo 2

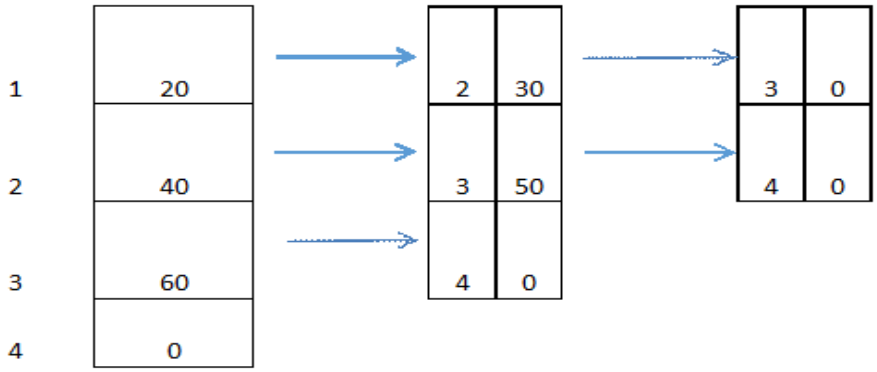
	A	B	C	D
A		1	1	
B	1		1	
C	1	1		1
D			1	

Nota: Se coloca un 1 en los respectivos lados adyacentes de un lado dado así: adyacentes a lado A los lados B y C por lo que esos cruces en la matriz tienen asignado 1 respectivamente.

- **Listas ligadas de adyacencia:** Se representa utilizando una lista simplemente ligada por cada vértice que enlaza con la dirección de los nodos adyacentes a él. Los apuntadores de entrada a cada vértice se encuentran en un vector de apuntadores de entrada. La configuración del nodo de la representación es:

Vértice	Liga

La representación con listas ligadas de adyacencia para el Grafo1 será:



- **Multilista de adyacencia:** Se define un registro para representar cada lado del grafo. El lado está conformado por dos vértices. La configuración del nodo es la siguiente:

V_i	V_j	LV_i	$L V_j$	Sw

LV_i: Apunta hacia otro registro que representa un lado incidente a V_i

LV_j: Apunta hacia otro registro que representa un lado incidente a V_j

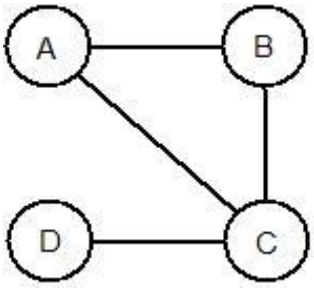
Sw: bandera usada para métodos sobre el grafo.

Donde se crea un vector V[i] apunta al primer nodo de la lista de nodos con los que se representan los lados incidentes al vértice i. La lista ligada que corresponde a un vértice v contiene los lados incidentes sobre el vértice v

- **Matriz de incidencia:** se define como una matriz de m filas y n columnas con: n: número de vértices del grafo y n: número de lados del grafo.

  Para hacer la representación se deben numerar los lados del grafo (aleatoriamente).

  Para el grafo 2 del ejemplo inicial:



Lado (A,B) se numera con 1

Lado (A,C) se numera con 2

Lado (B,C) se numera con 3

Lado (C,D) se numera con 4

En este caso m=4, n=4 (la matriz tiene 4 filas representan los vértices y 4 columnas representan los lados)

Si denominados la matriz como In

In[i][j] es igual a 1 si el lado j es incidente sobre el vértice i y es 0 si el lado j no es incidente sobre el vértice i

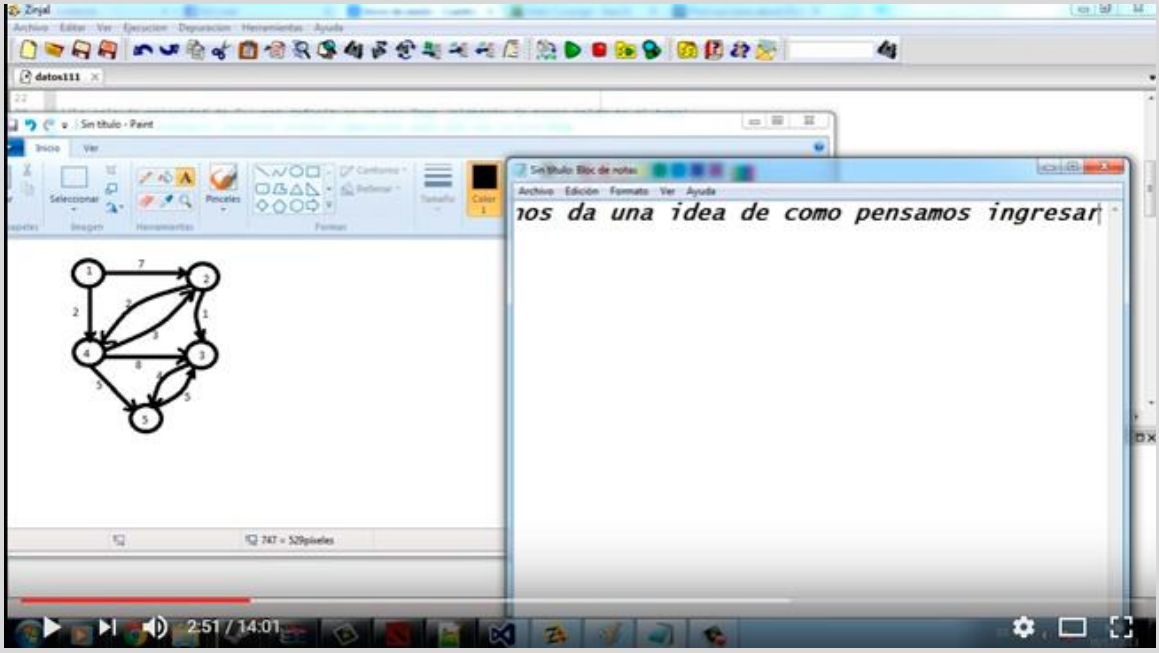
La representación del grafo seria:

In	1	2	3	4
A	1	1		
B	1		1	
C			1	1
D				1

¿Cuál **representación** de las propuestas para grafos se debe usar?

Todo **depende del problema** que se esté intentando resolver y además se debe **acomodar** a las **intenciones de diseño** del programador.

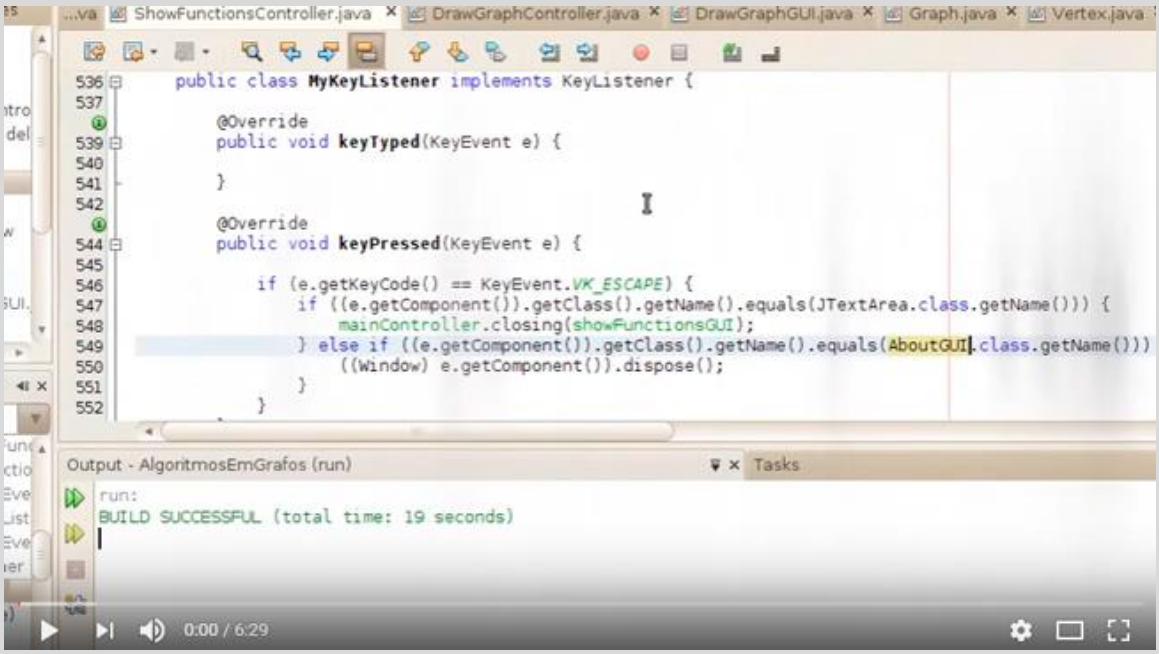
Para crear un grafo en programación a partir de su representación que se recomienda y aplicación de un proyecto de grafos en una situación real:



GRAFOS c++ distancias mas cortas Enlace

Proyecto Final Estructura de Datos (grafos) Enlace

Además sería muy bueno ver representación de grafos en java:



Representação Gráfica de um Grafo em Java V. 3.2 Enlace

3.2.1.4 RECORRIDOS SOBRE GRAFOS:

El principio básico para el manejo algorítmico de la estructura grafos se fundamenta en sus dos recorridos

- **Recorrido DFS**: Es la sigla en inglés que quiere decir primero búsqueda en profundidad. Este algoritmo funciona ubicándose en un vértice cualquiera del grafo y determina los vértices adyacentes a este y escoge uno que aún no haya sido visitado y a partir de ahí realizar un llamado recursivo al mismo algoritmo. El algoritmo DFS requiere controlar cuales vértices han sido visitados y cuáles no.

  Para ejercer el control sobre los visitados o no se utiliza un vector v[i] que tiene 0 si el vértice I no ha sido visitado y que vale 1 si el vértice ya fue visitado. Inicialmente el vector de visitados se inicializa con ceros.

  Un algoritmo para el DFS seria:

```
Void DFS ( int i )                      //el  método recibe el vértice donde arranca el grafo
        V[i]=1                          // el vértice ya fue visitado se pone en uno el vector
        Para todo vertice w adyacente a i haga  //ciclo que recorre los vértices adyacentes
                If (V[i]==0) entonces   // pregunta si el vértice no ha sido visitado
                        DFS(w)          // hace un llamado recursivo a DFS con w
                Fin (if)
        Fin (para)
Fin (DFS)
```

  El parámetro i indica el vértice a partir del cual se comienza a hacer el recorrido DFS. La determinación de los vértices adyacentes depende de la forma en que está representado el grafo.

  Un método general de representación para un grafo teniéndolo representado como listas ligadas de adyacencia seria:

```
Void DFS (entero i)                     //el  método recibe el vértice donde arranca el grafo
        V[i]=1                          // el vértice ya fue visitado se pone en uno el vector
        p=vec[i]                        // p apunta al primer nodo del vector de la lista
        While (p<>null) do              // recorre con p hasta el ultimo nodo
                w=p.retornaDato()       // asigna a w el dato de p
                If(V[w]==0) entonces    //  pregunta si w fue visitado o no
                        DFS(w)          // si no fue visitado hace llamado recursive a DFS
                Fin(if)
                P=p.retornaLiga()       // Actualiza la liga de p
        Fin(mientras)
Fin (DFS)
```

- **Recorrido BFS:**

  Su nombre en inglés lo que hace es justificar primero la búsqueda a lo ancho del grafo. En este algoritmo se visitan todos los vértices adyacentes a un vértice dado.
  La forma en que este recorrido se maneja sobre el grafo, supone que debe manejar una cola que indique el orden en que se han ido visitando los vértices, a este vector cola lo denominaremos visitado

El algoritmo para hacer un recorrido BFS sobre grafos en forma general es:

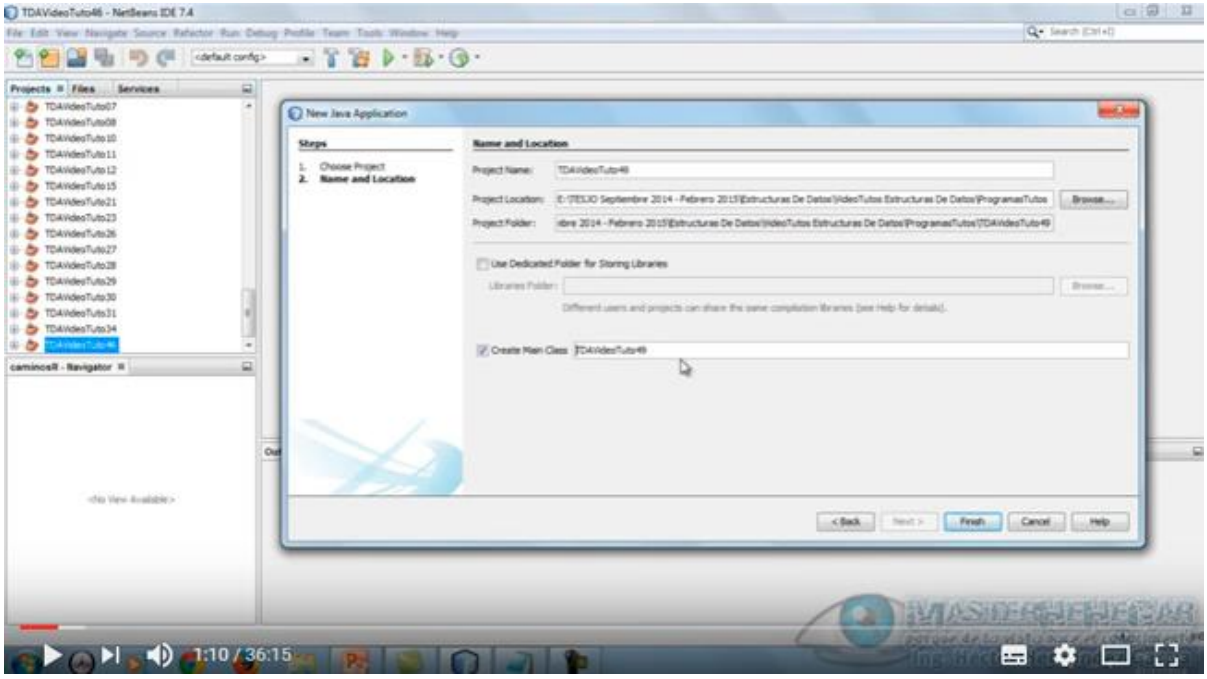
```
void BFS(entero v)                                  //recibe como parámetro el vértice donde inicia
        visitado[v]=1                               // marca el vértice como visitado
        cola.encolar(v)                             // encola el vértice en los visitados
        while ( cola.esvacia<>vacia) do             // ciclo para preguntar por  visitados
                v=cola.desencolar()                 //  desencola el vértice v
                imprima(v)                          //  imprime el vértice en el recorrido
                para todo vértice w adyacente a v haga // ciclo para recorrer vértices adyacentes a v
                        if(visitado(w)==0) then     // pregunta si w fue visitado o no
                                visitado(w)=1       // marca como visitado a w
                                cola.encolar(w)     // lleva el vértice a la cola
                        fis(si)
                fin(para)
        fin(while)
fin(BFS)
```

La determinación de los vértices w adyacentes a un vértice v dependerá de la forma como se tenga representado el grafo.

A continuación se muestra un recorrido BFS con el grafo representado como matriz de adyacencia:

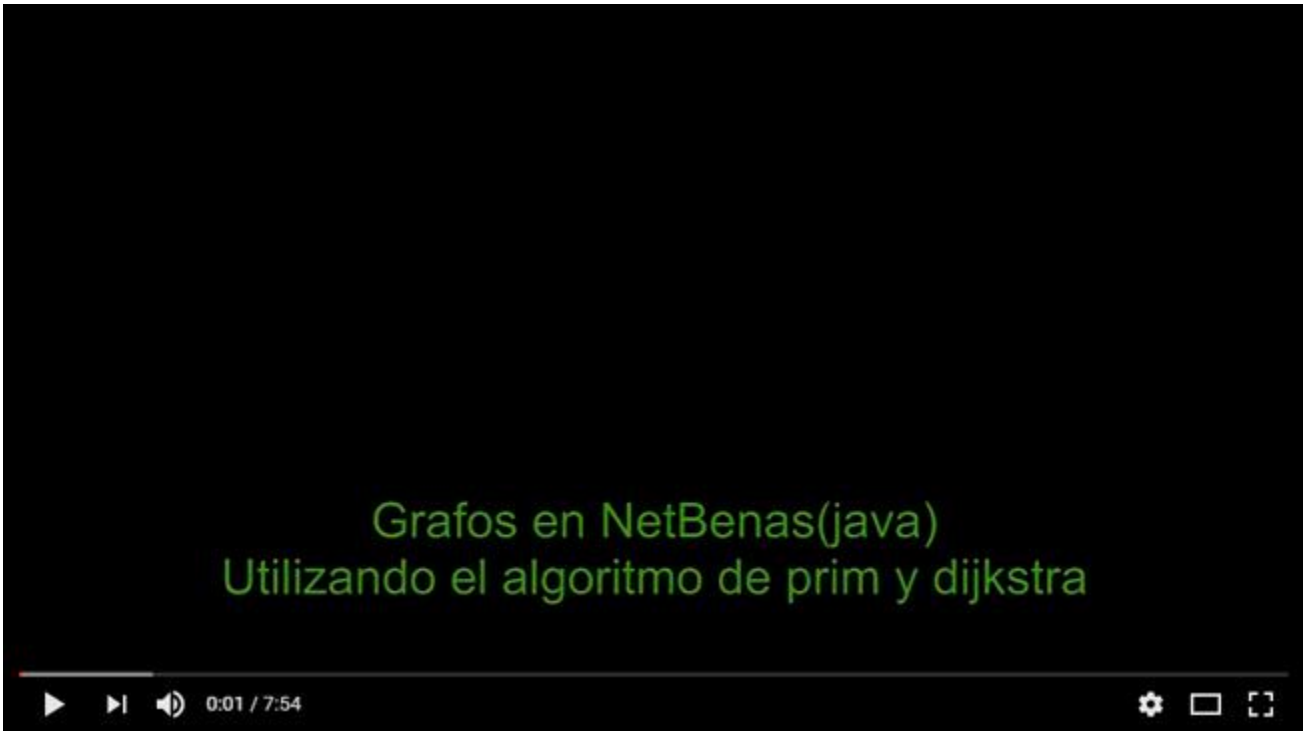
```
void BFS(entero v)                                  //recibe como parámetro el vértice donde inicia
        visitado[v]=1                               // marca el vértice como visitado
        cola.encolar(v)                             // encola el vértice en los visitados
        while ( cola.esvacia<>vacia) do             // ciclo para preguntar por  visitados
                v=cola.desencolar()                 //  desencola el vértice v
                imprima(v)                          //  imprime el vértice en el recorrido
                para (w=1;w<=n;w++) haga            // ciclo para recorrer adyacentes
                        if(adya(v)(w)==1) then      // pregunta por vector de adyacentes
                                if(visitado(w)=0) then // pregunta si w fue visitado o no
                                        visitado(w)=1 // marca w como visitado
                                        cola.encolar(w) // encola el visitado
                                fin(si)
                        fis(si)
                fin(para)
        fin(while)
fin(BFS)
```

Revisar implementación de grafos en Java Caminos mínimos:



49 - Grafos, El Camino Más Corto, Implementación (EDDJava) Enlace

Revisar grafos en Netbeans también interesante desde la programación con grafos



Grafos(Graphs) En NetBeans(java). prim y dijkstra Enlace

3.2.2 EJERCICIOS PROPUESTOS

1) Elabore un algoritmo que imprima el recorrido DFS de un grafo representado como matriz de adyacencia.

2) Elabore un algoritmo que imprima el recorrido BFS de un grafo representado como listas ligadas de adyacencia

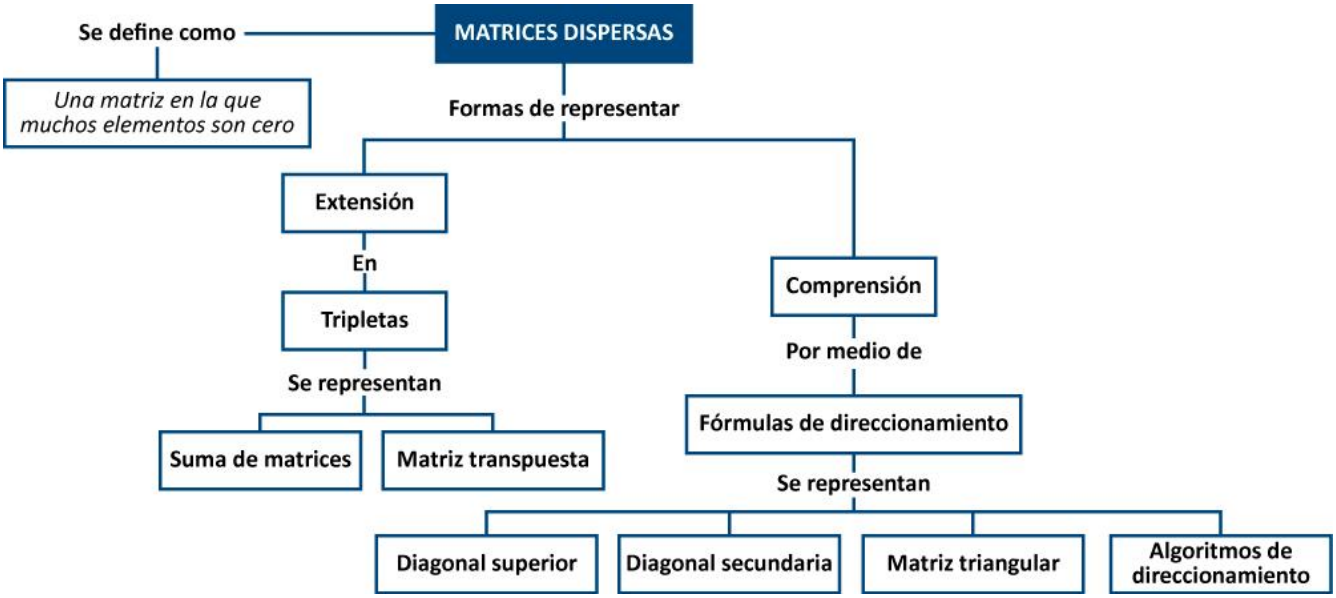
3) Elabore un algoritmo que imprima el recorrido BFS de un grafo representado como multilistas de adyacencia

4) Elabore un algoritmo que imprima el recorrido DFS de un grafo representado como multilistas de adyacencia

5) Elabore un algoritmo que imprima el recorrido BFS de un grafo representado como matriz de incidencia.

3.3 MATRICES DISPERSAS

Una aplicación de matrices que poseen muchos de sus elementos en ceros se muestra en las estructuras de datos como las matrices dispersas es algo que normalmente puede rebajar el procesamiento de algoritmos relacionados con métodos aplicados a este tipo de matrices.



3.3.1 RELACIÓN DE CONCEPTOS

Matriz dispersa: Es una matriz cuya propiedad es tener mayor cantidad de elementos en cero

Tripletas: forma de representar la matriz dispersa que consiste en fila, columna y valor

Direccionamiento: Formula para ubicar los elementos o datos de una matriz dispersa

Transpuesta de una Matriz: Es en la que se cambian filas por columnas y columnas por filas

Matriz triangular superior: los elementos situados debajo de la diagonal principal son ceros

Matriz triangular inferior: Los elementos situados encima de la diagonal principal son ceros

3.3.2 DEFINICIÓN DE MATRICES DISPERSAS:

Se dice que una matriz es dispersa cuando muchos de sus elementos son ceros, por ejemplo la siguiente matriz**:**

M	1	2	3	4	5
1	0	0	-2	2	0
2	0	0	0	5	0
3	3	4	0	1	2
4	0	0	2	0	1
5	0	0	0	2	0

La matriz denominada M se le conoce como matriz dispersa pues tiene la mayor cantidad de valores en la matriz como ceros (se denomina así de manera intuitiva).

Existen dos formas para representar matrices dispersas, por extensión y por comprensión. por extensión usando tripletas, y por comprensión utilizando fórmulas de direccionamiento.

3.3.3 REPRESENTACIÓN DE MATRICES EN TRIPLETAS:

Cada elemento de la matriz está definido por sus posiciones i,j las cuales representan la fila y la columna respectivamente y para referenciar el elemento debemos escribir la fila, la columna y el valor. Podemos almacenar la matriz como una lista de tripletas i, j, valor.

La tripleta en su posición cero define el orden de la matriz y el número de elementos diferentes de cero. Para definir objetualmente el problema se debe definir: Una clase matriz que retorna un objeto de la clase tripleta. Otra clase denominada matriz en tripletas.

La representación gráfica de matriz en tripletas es:

0	5	5	10
1	1	3	2
2	1	4	-2
3	2	4	5
4	3	1	3
5	3	2	4
6	3	4	1
7	3	5	2
8	4	3	2
9	4	5	1
10	5	4	2

Definición de algoritmos orientados a objetos con las matrices dispersas representadas en tripletas

- Clase tripleta

  Esta es la representación de la clase tripleta

```
Clase tripleta
Privado:                          //la parte privada define las características
        Entero fila,columna       // principales de la clase y sus tipos asociados
        Objeto valor
```

```
Public:
        Tripleta(entero f, entero,c,objeto, v)   //constructor de la clase
        Void asignafila(entero f)                // determina la fila
        Void asignacolumna(entero c)             // determina la columna
        Void asignavalor(objeto v)               // asigna valor a la posición de la matriz
        Entero retornafila()                     // devuelve la fila
        Entero retornacolumna()                  // devuelve la columna
        Objeto retornavalor()                    // devuelve el valor de la posición de la matriz
Fin(tripleta)
```

Los algoritmos para dichos métodos son en la clase tripleta son:

```
Tripleta(entero f, entero c, objeto v)
        Fila=f
        Columna=c
        Valor=v
Fin (tripleta)

Void asignafila(entero f)
        Fila=f
Fin (asignarfila)


Void asignacolumna(entero f)
        Columna=c
Fin (asignarcolumna)


Void asignavalor(objeto v)

        Valor=v

Fin (asigna valor)




Void retornafila()

        Return fila                 //devuelve el valor de la fila

Fin (retornafila)

Void retornacolumna()

        Return columna              //devuelve el valor de la columna
```

```
Fin (retornacolumna)

Objeto retornaValor()

        Return(valor)                   //retorna el valor de la matriz

Fin (retornavalor)
```

Por ser una definición bastante simple de los algoritmos no requiere que sea explicada. A continuación se define la clase matizentripletas:

```
Clase  matrizentripletas
Privado
        Tripleta v[]                                    //característica privada de la clase
Publico:
        Matrizentripletas(tripleta t)                   //constructor
        Void asignanumerotripletas(entero n)
        Void asignatripleta(tripleta tx, entero i)
        Entero retornafilas()
        Entero retornacolumnas()
        Entero retornanumerotripletas()
        Entero retornatripleta(entero i)
        Void muestramatrizentripletas(tripleta tx)      //ver matriz en tripletas
        Void insertatripleta(tripleta tx)               //insertar tripletas
        Matriz en tripletas suma(matizentripletas,b)    //sumar matrices en tripletas
        Matriz en tripletas multiplica(matizentripletas,b)//multiplica matriz en tripletas
        Matriz en tripletas traspuesta(matizentripletas,b)//matriz transpuesta en tripletas
Fin(matrizentripletas)
```

Después de definir la clase se escriben los métodos básicos para la creación y manipulación de la representación en tripletas para la matriz dispersa así:

```
Matriz en tripletas construyematriz(entero m, entero n)     // m, n enteros para fila y columna de matriz
        Matriztripletas a                                   //define el objeto "a" de la clase
        Entero c,d                                          //define dos variables enteras
        t=new tripleta(n,m,null)                            //asigna a T memoria de tripleta
        a=new matrizentripletas(t)                          //asigna a a memoria de matriztripleta
        c=0                                                 //inicializa  "c "en cero
        For(i=1;i<=m;i++) do                                //ciclo para las filas
                For (j=1;j<=n;j++) do                       //ciclo para las columnas
                        d=retornadato(i,j)                  //devuelve el dato en i, j
                        If (d!=0) then                      //pregunta si d no es 0
                                c=c+1                       //cuenta en c la tripleta
                                t=new tripleta(i,j,d)       //asigna memoria de tripleta en "T"
```

```
                    a.asignatripleta(t,c)          //asigna el valor a la tripeta t,c
                End(if)
            End(for)
        a.asignadatos(c )                          //asigna datos a la matriz con asignadatos
        return a                                   //retorna la matriz en tripletas
fin (construyematrizentripletas)
```

El siguiente paso es construir la matriz en cuadricula cuando me dan la matriz en tripletas:

```
matriz contruyematriz()                            //identificación del método de la clase
        entero m,n,p                               //definición de tres variables enteras
        tripleta t                                 //definición de objeto de clase tripleta
        matriz a                                   //definición de objeto de clase matriz
        m=retornafilas()                           //número de filas
        n=retornacolumnas()                        //número de columnas
        p=retornanumerodetripletas()               //número de tripletas
        a=new matriz(m,n)                          //pide memoria para objeto a de matriz
        for (i=1;i<=p;i++) do                      //ciclo para recorrer hasta número tripletas
            t=retornatripleta(i)                   //retorna tripleta t de acuerdo a i
            a.asignadato(t.retornafila(),t.retornacolumna(), t.retornavalor()) //crea la tripleta
        end(for)
        fin(construyematriz)
```

A continuación se escribirá el método matriz en tripletas

```
        matrizentripletas(tripleta t)          //constructor de la clase
            entero m=t.retornafila()           //encuentra las filas de la matriz
            entero n=t.retornacolumna()        //encuentra las columnas de la matriz
            entero p=m*n+2                     //determina la cantidad de elementos de la matriz
            entero I                           //define variable entera
```

```
v=new tripleta[p]                    // asigna memoria de clase tripleta a "v"
v[0]=t                               //inicializa el vector de tripletas en el valor de "t"
for (i=1,i<=p,i++) do                //ciclo que recorre con variable "i" hasta elementos "p"
        v[i]=null                    //asigna cero a las posiciones de la matriz
end(for)
fin(matrizentripletas)
```

El constructor recibe como parámetros una tripleta t, la cual tiene las dimensiones de la matriz a representar en la fila y columna y en el campo de valor tiene asignado un cero (indica matriz vacía). Se define un vector de tamaño n*m+2, para tener una tripleta adicional que facilita algunos de los algoritmos siguientes. La tripleta de posición cero almacena las dimensiones de la matriz y el número de elementos diferentes de cero.

Los métodos más importantes asociados a tripleta son:

```
void asignatripleta(tripleta tx, entero i)
        v[i]=tx
fin (asigna tripleta)
```

El método asigna la tripleta enviada al vector v en la posición i

```
Void asignanumerode tripletas(entero n)
        tripleta t=V[0]
        t.asignavalor(n)
        v[0]=t
fin (asignanumerodetripletas)
```

El método actualiza el campo de valor de la tripleta que se halla en la posición 0 del vector v (indica el número de elementos que se representan)

```
Entero retornafilas()
        Tripleta t=v[0]
        Return t.retornafila()
```

```
Fin(retornafilas)

Método que Devuelve las filas de la matriz

Entero retornacolumnas()
        Tripleta t=v[0]
        Return t.retornacolumnas()
Fin(retornacolumnas)

Método que devuelve las columnas de la matriz

Entero retornanumerotripletas()
        Tripleta t=v[0]
        Return t.retornavalor()
Fin(retornanumerotripletas)

Método que devuelve el número de elementos diferentes de cero.

Entero retornatripleta(entero i)
        Return v[i]
Fin(retornatripleta)

Retorna la tripleta que se halla en la posición i del vector v

Void muestramatrizentripletas()
        Entero i=1
        Tripleta t =retornatripleta(0)
        Entero datos=t.retornavalor()
        While(i<0datos)
                Imprima(v[i].retornafila(),v[i].retornacolumna(),v[i].retornavalor())
                I=i+1
        End(while)
```

```
Fin(muestramatrizentripletas)
```

El anterior método simplemente recorre el vector de tripletas escribiendo los datos contenidos en cada tripleta.

```
void insertatripleta(tripleta i)                          //método para insertar tripletas
        Entero i,j,datos                                  //define tres variables enteras
        Tripleta t, tx                                    //define objetos de tripleta
        tx=retornatripleta(0)                             //asigna en tx tripleta cero
        datos= tx.retornavalor()                          //retorna valor de tripleta cero
        i=1                                               //inicializa control de ciclo
        t=retornatripleta(i)                              //retorna tripleta 1 en t
        While (i<=datos and t.retornafila() <ti.retornafila()) do  //
                i=i+1
                t=retornatripleta(i)
        end(while)
        datos =datos+1
        j=datos-1
        while(j>=i) do
                v[j+1]=v[j]
                j=j-1
        end(while)
        v[i]=ti
        Asignanumerotripletas(datos)
Fin (insertatripleta)
```

El parámetro de entrada es la tripleta que se desea insertar

Explicación del método insertar:

1. Buscamos la primera tripleta que contenga la fila de la tripleta ti

2. Cuando se avanza sobre el vector de tripletas se tiene en cuenta que estemos en la misma fila de la tripleta ti y la columna de la tripleta ti sea mayor o igual que la de la tripleta i, en los dos ciclos el valor de i debe ser menor o igual que el número de tripletas de la matriz. Al finalizar los dos ciclos la variable i queda apuntando hacia la posición donde se debe insertar la nueva tripleta. Lo último es su ubicación dentro del vector y correr los datos si es necesario en el vector.

3. El orden de magnitud es O(p), siendo p el número de tripletas (orden lineal)

- A continuación se presenta un método para calcular la transpuesta de una matriz dispersa representada en tripletas:

La traspuesta es la matriz que resulta de intercambiar las filas con las columnas en una matriz (en otras palabras la fila i será la columna i y la columna j será la fila j. De otra forma un elemento de la fila i columna j en la matriz original quedara en la fila j columna i dentro de la matriz transpuesta.

```
Matrizentripletas transpuesta                          //método que genera la transpuesta de matriz
    Entero i, p ,f , c, v                              //define cinco variables enteras
    Tripleta ti                                        //define objeto ti de clase tripleta
    p =retornanumerotripletas()                        //asigna a p el número de tripletas
    ti=new tripleta(retornacolumnas(), retornafilas(),0)   //asigna a ti memoria de tripleta
    matriz entripletas b=new matrizentriplets(ti)      //asigna a b memoria de matrizentripletas
    i=1                                                //inicializa i en 1
    while(i<=p) do                                     //ciclo mientras hasta el número de tripletas
        ti=retornatrileta(i)                           //retorna tripleta en ti
        f=ti.retornacolumna()                          //determina la fila de la matriz
        c=ti.retornafila()                             //determina la columna
        v=ti.retornavalor()                            //determina el valor
        ti=new tripleta(f,c,v)                         //asigna memoria de tripleta en ti para c, f
        b.insertatripleta(ti)                          //inseta tripleta en el objeto b (matriz transp)
```

```
        i=i+1                                   //incrementa el valor de i
    end(while)
    return b                                    //retorna b con matriz transpuesta en tripletas
fin(transpuesta)
```

Es un método que se basa en llamar a insertatripleta para crear la transpuesta. El orden de magnitud de este algoritmo es $O(p^2)$, pero en el peor de los casos con el llamado a insertatripleta puede ser $O(m^2*n^2)$ que sería un algoritmo catastrófico.

Para evitar el llamado inserta tripleta dentro del algoritmo de la transpuesta en tripletas se propone el siguiente algoritmo para calcular la transpuesta en tripletas:

- Variación del algoritmo de la transpuesta en tripletas para mejorar su rendimiento

```
MatrizentripletasTranspuestaM()
    Entero i,j,k,m,n,p,f,c,v
    Tripleta tj, tx
    m=retornafilas()
    n=retornacolumnas()
    p=retornanumerotripletas()
    tx= new tripleta(n,m,p)
    matrizentripletas b=new matriz entripletas(tx)
    k=0
    for(i=1;i<=n;i++) do
        for(j=1;j<=p;j++) do
            tj=retornatripleta(j)
            f=tj.retornafila()
            c=tj.retornacolumna()
            v=tj.retornavalor()
```

```
                    if (c==i) then
                        k=k+1
                        tx=new  tripleta(c,f,v)
                        b.asignatripleta(tx,k)
                    end(if)
                end(for)
            end(for)
            return b
fin(traspuestaM)
```

El anterior algoritmo de la transpesta elimina el llamado a insertatripleta, pero hacienda un análisis al algoritmo el orden de magnitude puede llegar a ser $O(m*n^2)$ el cual comparado al algoritmo de la traspuesta en cuadriculas $O(m*n)$ sigue siendo malo.

- Una tercera alternativa de algoritmo de la traspuesta en tripletas es:

```
Matrizentripletas transpuestar()
        entero m,n,p,i,j,s[],t[]
        tripleta  ti, tx
        m=retornafilas()
        n=retornacolumnas()
        p=numerodetripletas()
        ti=new tripleta(n,m,p)
        matrizentripletas b=new matriz entripletas(ti)
        s=new entero[n+1]
        t= new entero[n+1]
        for(i=1;;i<=n;i++) do
            s[i]=0
```

```
        end (for)
        for(i=1;;i<=p;i++) do
                ti=retornatripleta(i)
                s[ti.retornacolumna()]=s[ti.retornacolumna()]+1
        end (for)
        t[1]=1
        for(i=2;i<=n;i++)
                t[i]=t[i-1]+s[i-1]
        end(for)
        for(i=1;;i<=p;i++) do
                ti=retornatripleta(i)
                j=ti.retornacolumna()
                tx= new tripleta(j, ti.retornafila(), ti.retornavalor())
                b.asignatripleta(tx,t(j))
                t[j]=t[j] +1
         end (for)
        return b
fin(transpuestar)
```

El algoritmo anterior usa dos vectores para realizar dentro del método menos repeticiones en las instrucciones de los ciclos, lo cual mejora el rendimiento del algoritmo.

Si aplicamos análisis de algoritmos a esta última propuesta el orden de magnitud es O(m+n) y en el peor de los casos seria: =O(m*n), que en la mayoría de los casos mejora la eficiencia de un algoritmo con cuadriculas.

4.2 Representación de matrices dispersas con fórmulas de direccionamiento

4.2.1 Representación de matrices diagonales en vectores con fórmulas de direccionamiento

Sea una matriz cuadrada de orden n*n, donde todos los elementos de la diagonal principal son distintos de cero así como el siguiente ejemplo:

f/c	1	2	3	4	5	6
1	10					
2		2				
3			5			
4				38		
5					20	
6						16

n=6

Si representamos la matriz en forma tradicional gastamos n^2 posiciones de memoria, de los cuales se usan solamente n, para ahorrar memoria debemos de utilizar un vector de n posiciones para almacenar los datos de la diagonal principal así:

1	2	3	4	5	6
10	2	5	38	20	16

La representación anterior queda como la de un vector con tamaño igual a la cantidad de elementos diferentes de cero de la matriz dispersa diagonal principal.

La fórmula de direccionamiento es:

pos =i o pos =j y es válida para los elementos de la matriz m[i][j] que cumplan que i=j (se debe decir para cual rango de filas y columnas es válida dicha fórmula)

4.1.3.2 Matriz de diagonal secundaria

Sea la matriz que representa los elementos de la diagonal secundaria:

f/c	1	2	3	4	5	6
1						8
2					20	
3				10		
4			7			
5		5				
6	2					

Se podría representar usando un vector para almacenar los datos de la diagonal secundaria así:

1	2	3	4	5	6
8	20	10	7	5	2

La fórmula de direccionamiento Pos =i si hacemos la representación por filas

Si hacemos la representación por columnas se obtiene:

1	2	3	4	5	6
2	5	7	10	20	8

Y la fórmula de direccionamiento es:

Pos=j

Se debe tener en cuenta que para pertenecer a la diagonal secundaria un elemento ubicado en i, j de la matriz debe cumplir que Para todo **i, j : i + j=n+1**

3.4 ANÁLISIS DE OTRAS FÓRMULAS DE DIRECCIONAMIENTO

3.4.1 FORMULA DE DIRECCIONAMIENTO DE MATRIZ TRIANGULAR INFERIOR IZQUIERDA

Este tipo de matrices aparecen regularmente en soluciones del algebra y programación lineal. Una matriz de este tipo con grandes dimensiones tiene más de la mitad de sus elementos en cero por eso se hace indispensable representar este tipo de matrices como vectores con todos sus elementos diferentes de cero sacados de la matriz dispersa. La siguiente es la representación de las matrices que tienen un comportamiento disperso como triangular:

Se representa en el caso de 3*3 de la siguiente manera:

f/c	1	2	3
1	4		
2	9	7	
3	5	3	10

La representación de la matriz triangular inferior izquierda como vector por filas seria:

1	2	3	4	5	6
4	9	7	5	3	10

y en general el comportamiento del vector que se genera es:

En la fila 1 tiene un solo dato diferente de cero de la matriz

En la fila 2 tiene dos datos diferentes de cero de la matriz

En la fila 3 tiene tres datos diferentes cero de la matriz

En general en la fila i, tiene i datos diferentes de cero de la matriz. Teniendo en cuenta el la progresión aritmética de datos diferentes de cero por fila seria: 1,2,3,4,……i

Para calcular la sumatoria de esta progresión aplicamos la fórmula matemática para la suma de una progresión aritmética así:

Suma= $\sum_1^n i=\frac{n(n+1)}{2}$

En todas las matrices de este tipo se analiza: 1<=i<=n y 1<=j<=n , con i número de la fila y j número de la columna. Aplicando el método de inducción matemática en este caso se obtiene que un elemento perteneciente a la fila i columna j su pos el vector se obtiene con base a la suma de los elementos de las i-1 filas anteriores más la columna j así:

pos para datos diferentes de cero de la fila 1, columna 1 es: i-1+j=1-1+1=1

pos para datos diferentes de cero de la fila 2, columna 1 es: i-1+j=2-1+1=2

pos para datos diferentes de cero de la fila 3, columna 1 es: 3-1+1=3

En general pos =($\sum_1^{i-1} i$)+j

$\sum_1^{i-1} i$=i*(i-1)/2

De esa manera que:

Pos=i*(i-1)/2+j

En este caso es válido para los elementos que pertenezcan a la matriz triangular inferior, es decir:

Para todo i,j tales que i>=j

3.4.2 FORMULA DE DIRECCIONAMIENTO DE MATRIZ TRIDIAGONAL PRINCIPAL (ESTILO, TÍTULO 4)

La característica principal de esta matriz es que los únicos elementos diferentes de cero se encuentran en la diagonal principal y sus diagonales adyacentes. Ejemplo con una matriz dispersa de 4*4

f/c	1	2	3	4
1	8	5		
2	15	9	6	
3		11	2	10
4			20	1

Matriz tridiagonal

Su respectiva representación por filas de la matriz dispersa en un vector seria:

1	2	3	4	5	6	7	8	9	10
8	5	15	9	6	11	2	10	20	1

En este caso y por el mismo método de inducción matemática sugerido en el análisis de la anterior formula de direccionamiento se encuentra que:

pos=2*(i - 1) + j

Para todo i,j tal que valor absoluto(i-j)<2

Nota: el proceso inverso de determinar los subíndices i y j conocido pos y n y la formula de direccionamiento, basta con dividir la posición por 2 y sumarle 1 al cociente para obtener la fila i. Conocido i despejamos j usando la formula.

Otras fórmulas de direccionamiento con matrices dispersas se pueden encontrar en:

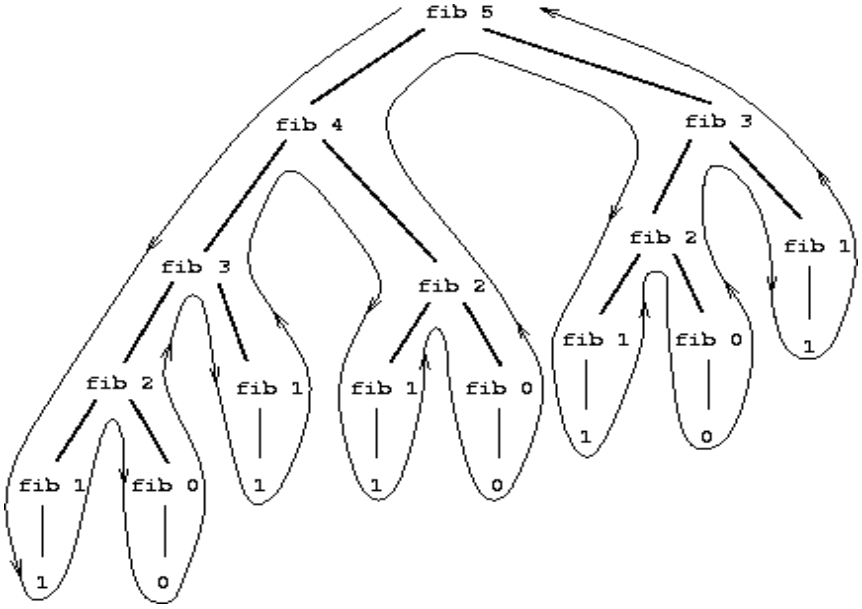
Estructuras de Datos para Matrices Dispersas.© UPV Enlace

Ejercicios propuestos

- Deducir la fórmula de direccionamiento de una matriz dispersa triangular inferior derecha para representar por filas en un vector
- Deducir la fórmula de direccionamiento de una matriz dispersa triangular superior derecha para representar por filas en un vector
- Deducir la fórmula de direccionamiento de una matriz dispersa triangular superior izquierda para representar por filas en un vector
- Deduzca una fórmula de direccionamiento para representar por filas en un vector una matriz tridiagonal secundaria

4 PISTAS DE APRENDIZAJE

Recuerde que: Un árbol general es un concepto recursivo por que el árbol se define en función del mismo (ósea llamado en su creación a otra estructura con las mismas características. Así gráficamente:



Un hijo de una raíz en un árbol puede ser otro árbol con las mismas características.

Tenga en cuenta que: Un árbol general no puede ser vacío, pero un árbol binario se define con: n>=0 registros ósea que ese si puede ser una estructura de datos vacía.

Recuerde que: La altura de un árbol binario depende del número de niveles que tiene el árbol (es el mayor nivel que este puede alcanzar). Para encontrar en java como calcular la altura de un árbol puedo remitirme al siguiente video en YouTube:

https://www.youtube.com/watch?v=_P0JRbqEEX0

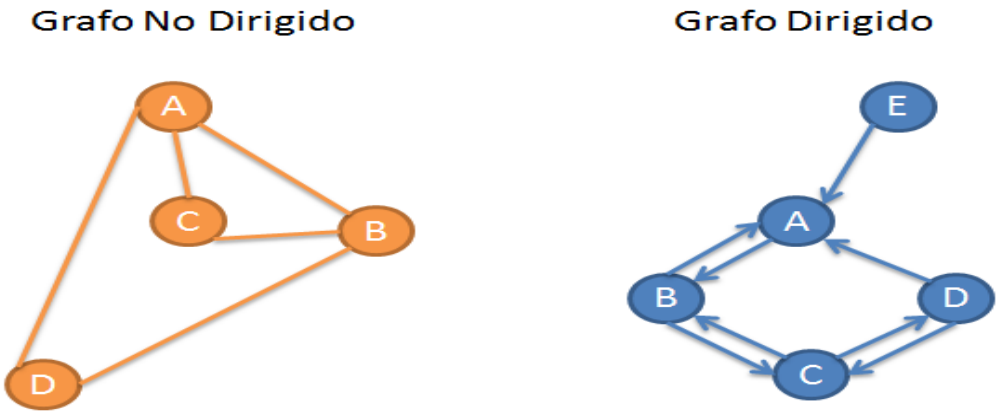
Tenga en cuenta que: Que el recorrido preorden visita primero la raíz (la imprime) y el recorrido postreden visita por último la raíz (la imprime)

No puede olvidar: Que un seguimiento recursivo sobre arboles debe guardar las direcciones de retorno necesarias para hacer todo su trabajo recursivo en una pila de direcciones vea el siguiente video:

https://www.youtube.com/watch?v=35Qdl0p3m6E

Recuerden por favor: que las representaciones que tienen en cuenta el registro de padre en sus nodos hijos pueden devolverse a sus respectivos padres cuando se requiera en el método de la clase trabajada.

Recuerde bien: La diferencia entre un grafo dirigido y uno no dirigido



El grafo dirigido marca la orientación exacta de los lados, el no dirigido tiene las dos orientaciones.

Recuerde por favor que los grafos se pueden utilizar en la realidad en:

Mas sobre esta respuesta de grafos: https://www.youtube.com/watch?v=-zpx_wMUyQg

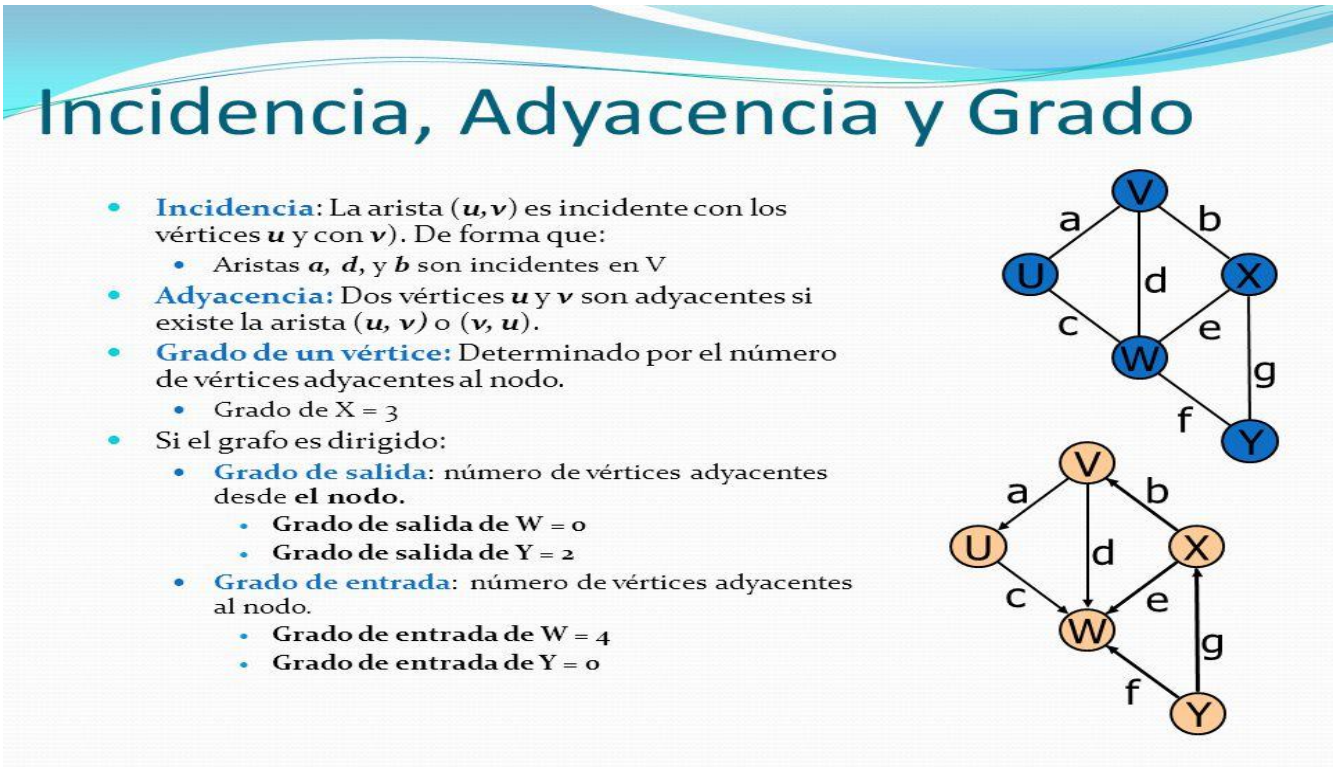
No olvide que: los grafos tiene representaciones dinámicas y estáticas. Ver el video siguiente en YouTube sobre el tema en la siguiente dirección: https://www.youtube.com/watch?v=cEfOVtaBcKw

Recuerde cual es: el concepto de lado, vértice, ciclo, camino, grado a nivel de un grafo cuando se habla de ruta o camino entre grafos:

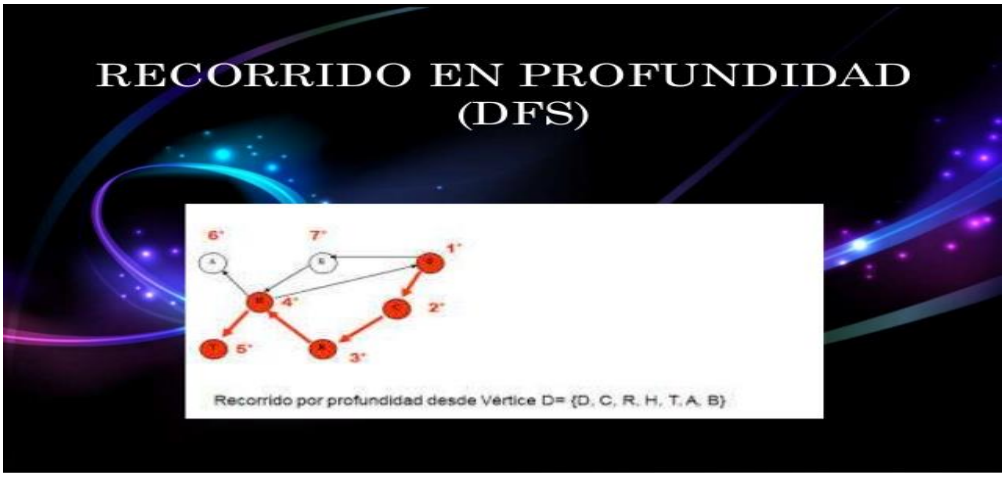
Ver enlace de internet con el pdf correspondiente:

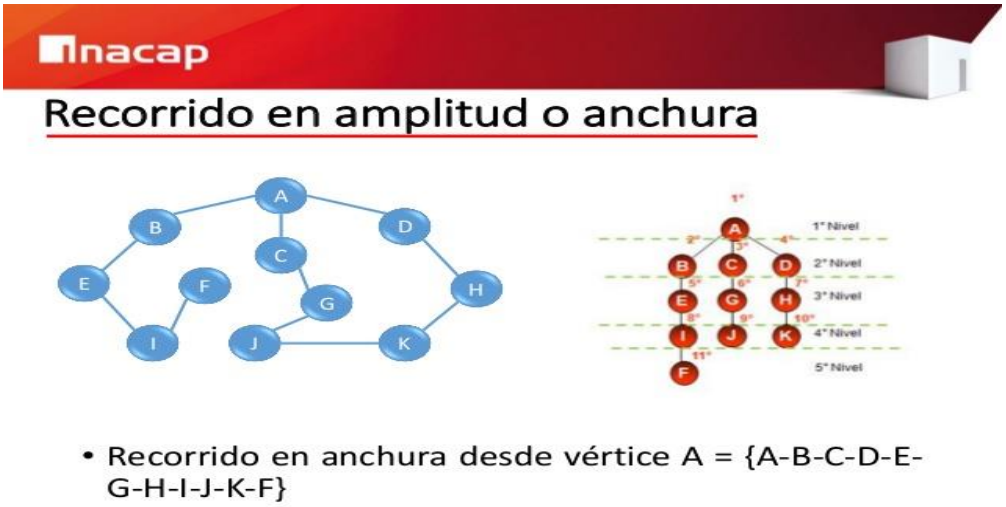
http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Estructura%20de%20Datos/Apuntes/grafos/Apuntes_Grafos.pdf

Recuerde: la diferencia ente incidencia y adyacencia en la teoría de grafos. Ver la imagen siguiente:



No olvide por favor: Las diferencias entre los recorrido en anchura y profundidad.





Para reforzar el conocimiento ver video en la siguiente dirección:

https://www.youtube.com/watch?v=Y4tndzfQkoE

Recuerde Bien: Cual es la importancia de las listas generalizadas a nivel de las estructuras de datos?

Leer el siguiente pdf y confrontar las diferencias: http://www.inf.udec.cl/~andrea/cursos/estructura/Listas.pdf

No olviden por favor sobre la importancia de las matrices dispersas en aplicaciones lineales y en matemáticas ver:

https://www.youtube.com/watch?v=kzJ9Ux2xwSI

Recuerden Muy bien lo importante que es optimizar el almacenamiento de memoria cuando se usa aun estructuras de datos

Como las matrices dispersas que tienen muchos datos en ceros. Es más conveniente aprender a realizar almacenamiento de los datos de la matriz en una estructura lineal (baja el orden de magnitud de los algoritmos optimizando código de programación)

Recuerde también: la representación estática con tripletas en Geogebra de una matriz dispersa:

https://www.youtube.com/watch?v=cicfbhRJKm0

5 BIBLIOGRAFÍA

Becerra, S. C. (2000). *Estructura de datos en java.* bogota: Kimpres limitada.

Florez, r. (2012). *Algoritmia 3.* Medellin: universidad de antioquia.

Gotieb, C. C. (1978). *Data type and structures.* New jersey: Prentice Hall.

Joyanes Aguilar, l. (1999). *Estructura de datos, libro de problemas.* Madrid: McGrawHill.

Marti, O. O. (2004). *Estructuras de datos y metodos algoritmicos.* Madrid : Prentice Hall.