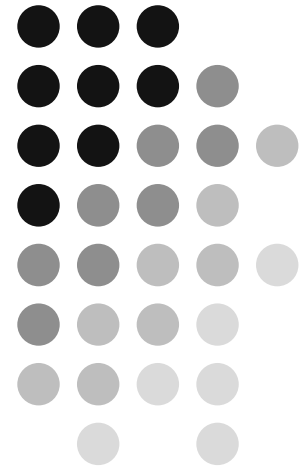


Algoritmos para grafos. Fundamentos

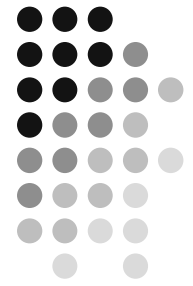


UNIVERSIDAD
DE LOS ANDES

Diseño y Análisis de Algoritmos
Cátedra de Programación
Carrera de Ingeniería de Sistemas
Prof. Isabel Besembel Carrera

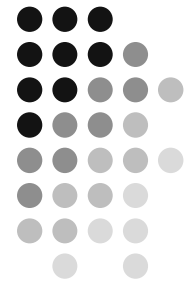


Introducción

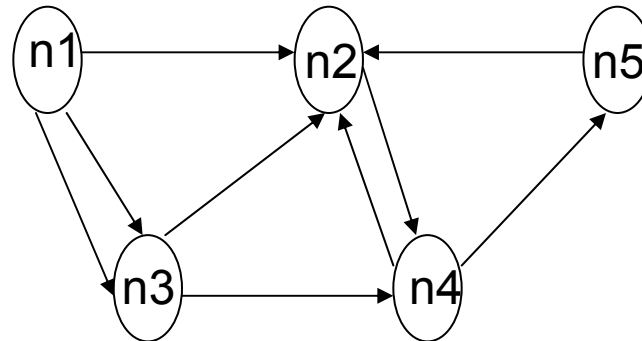


- Las estructura de datos no lineales se caracterizan por no existir una relación de adyacencia, entre sus elementos, es decir, un elemento puede estar relacionado con cero, uno o más elementos.
- Entre las múltiples aplicaciones que tienen estas estructuras podemos mencionar:
 - ❖ Los grafos son estructuras que se utilizan para modelar diversas situaciones tales como: sistemas de aeropuertos, flujo de tráfico,...
 - ❖ Los grafos también son utilizados para realizar planificaciones de actividades, tareas del computador, planificar operaciones en lenguaje de máquinas para minimizar tiempo de ejecución.

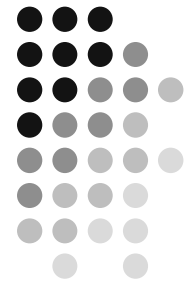
Introducción



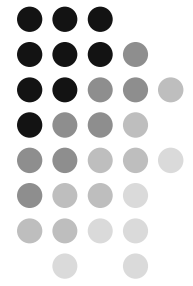
- La estructura no lineal de datos más general es el grafo, donde sus nodos pueden relacionarse de cualquier manera sin una relación de orden predefinida.
- Los grafos pueden ser utilizados como la estructura básica para múltiples aplicaciones en el área de la Computación



Definición



- Un grafo $G(N, A, f)$ es un conjunto no vacío de:
 - ❖ $N = \{n_1, n_2, \dots, n_M\}$ nodos o vértices,
 - ❖ $A = \{a_1, a_2, \dots, a_K\}$ aristas y
 - ❖ la función $f: R \rightarrow M \times M$ que indica los pares de nodos que están relacionados.
- Hay dos tipos de grafos, los dirigidos o digrafos y los no dirigidos, según que sus relaciones tengan dirección o no, respectivamente.
- La representación gráfica de un grafo se define con un círculo o rectángulo para los nodos y las relaciones con líneas o flechas según sea un grafo no dirigido o un digrafo, respectivamente

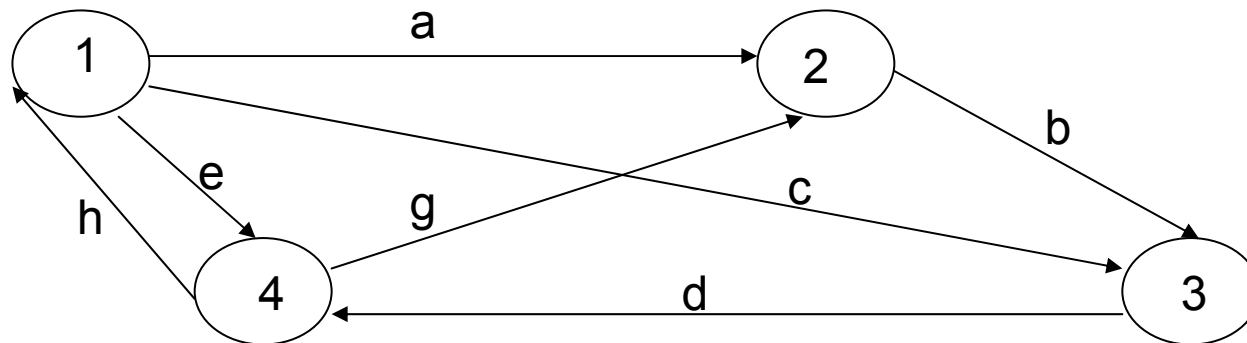


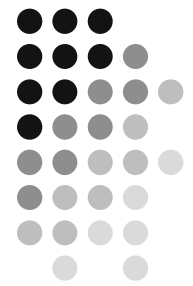
➤ **Digrafo**, donde $D = \{N, A, f\}$ tiene

$N = \{1, 2, 3, 4\}$,

$A = \{a, b, c, d, e, g, h\}$ y

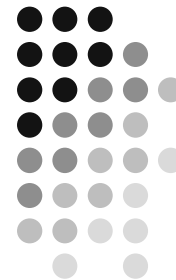
$f(a) = (1, 2)$, $f(b) = (2, 3)$, $f(c) = (1, 3)$, $f(d) = (3, 4)$,
 $f(e) = (1, 4)$, $f(g) = (4, 2)$, $f(h) = (4, 1)$.



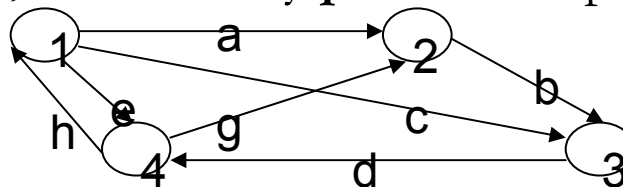


- Los elementos de un digrafo $D = \{N, A, f\}$ son el conjunto de nodos N , el conjunto de arcos A y la función $f : A \rightarrow N \times N$ de pares ordenados de nodos que indican de que nodo sale el arco y a que nodo llega ese arco.
- Los elementos de un grafo no dirigido $G = \{N, L, f\}$ son el conjunto de nodos N , el conjunto de líneas L y la función $f : L \rightarrow N \times N$ de pares no ordenados que indican que esos nodos están unidos en ambas direcciones.

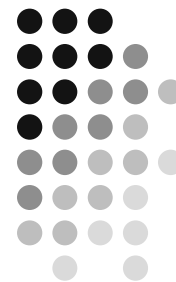
Conceptos básicos



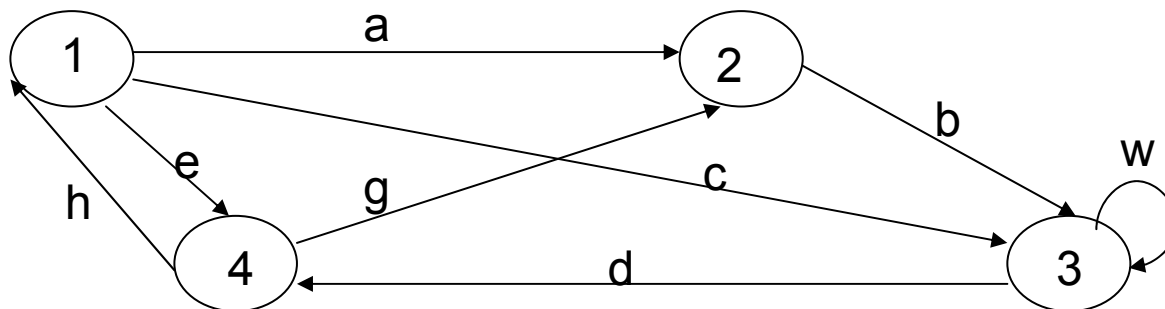
- Se dice que el $a_j \in A$ asociado al par de nodos (n_i, n_j) donde $n_i, n_j \in N$, **comienza** en n_i y **termina** en n_j . Así, n_i es el nodo **inicial** y n_j es el nodo **terminal**. Ejemplo: si $n_i = 1$, $n_j = 2$ y $a_j = \mathbf{a}$ en el digrafo D , entonces 1 es el nodo inicial de \mathbf{a} y 2 es el nodo terminal de \mathbf{a} .
- **Arco incidente:** a_j es un arco incidente sobre n_k , si n_k es el nodo terminal de a_j . Ejemplo: \mathbf{a} es el arco incidente sobre 2, pero no sobre 1.
- **Arcos adyacentes:** a_i y a_j son arcos adyacentes si a_i y a_j son incidentes en el mismo nodo. Ejemplo: \mathbf{c} y \mathbf{b} son arcos adyacentes, pero \mathbf{c} y \mathbf{d} no lo son.
- **Arcos paralelos:** Dos arcos son paralelos si ellos comienzan y terminan en los mismos nodos. Ejemplo: En D no hay arcos paralelos. Si anexamos a D el arco \mathbf{p} cuya $f(\mathbf{p}) = (1, 2)$, entonces \mathbf{a} y \mathbf{p} son arcos paralelos.



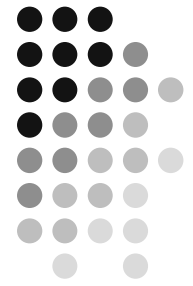
Conceptos básicos



- **Bucle o lazo:** Si un arco a_j comienza en n_i y termina en n_i , entonces a_j es un bucle. Ejemplo: Si $f(w) = (3, 3)$ entonces w es un lazo.
- **Nodos adyacentes:** Dados un par de nodos (n_i, n_j) que están unidos por el arco a_j , se dice que n_i es adyacente a n_j por a_j . Ejemplo: 1 es adyacente a 2 por **a**.

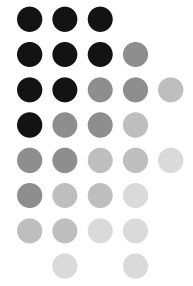


Conceptos básicos



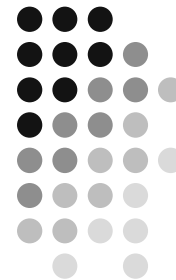
- **Grado de incidencia positivo:** El grado de incidencia positivo de un nodo n_j es el número de arcos que tienen como nodo inicial a n_j . Ejemplo: El grado de incidencia de 1 es igual a 3.
- **Grado de incidencia negativo:** El grado de incidencia negativo de un nodo n_j es el número de arcos que terminan en n_j . Ejemplo: El grado de incidencia negativo de 1 es igual a 1.
- **Grado de un nodo:** Para digrafos es el grado de incidencia positivo menos el grado de incidencia negativo del nodo. Ejemplo: El grado de 1 es igual a $3 - 1 = 2$, el grado del nodo 4 es $2 - 2 = 0$. Para grafos no dirigidos es el número de líneas asociadas al nodo.

Conceptos básicos

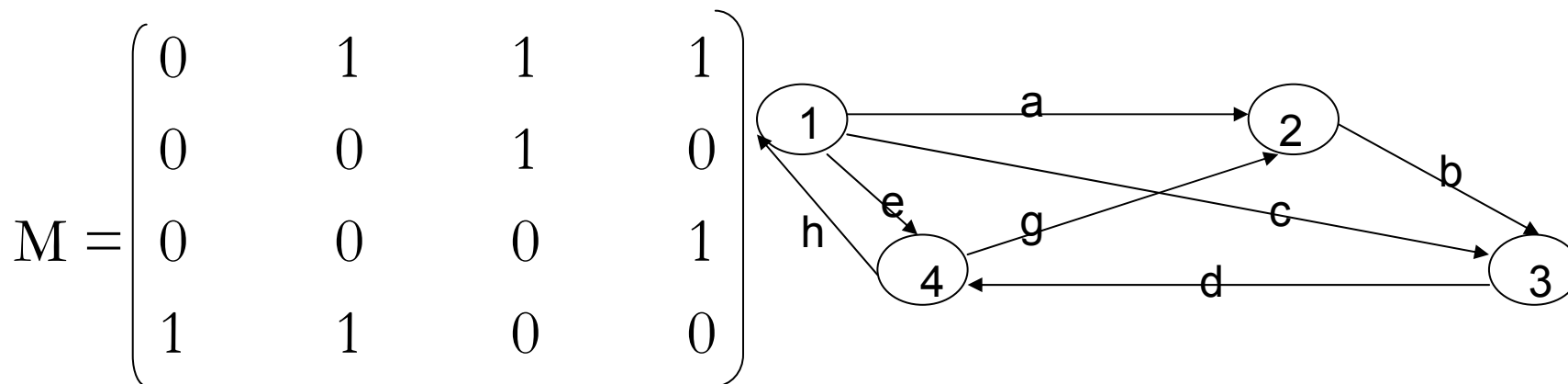


- **Multiplicidad de un par ordenado (n_i, n_j) :** Dado el digrafo D . Si para un par ordenado de nodos $(n_i, n_j) \in \mathbb{N} \times \mathbb{N}$, hay k arcos distintos en A para los que $f(a_j) = (n_i, n_j)$, es decir, hay k arcos que unen n_i con n_j , se dice que el par (n_i, n_j) tiene multiplicidad k donde $k \geq 0$. La función de multiplicidad $m(n_i, n_j) = k$. Ejemplo: $m(1, 1) = 0$, $m(1, 2) = 1$, $m(1, 3) = 1$, etc.
- **Grafo simple:** Un digrafo D es simple si cada par ordenado $(n_i, n_j) \in \mathbb{N} \times \mathbb{N}$ tiene como máximo de multiplicidad el valor 1. Ejemplo: El digrafo D de la figura es simple.

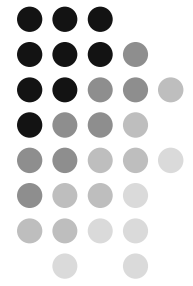
Conceptos básicos



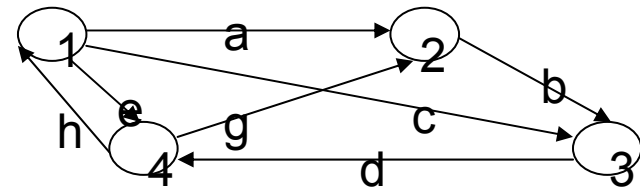
- **Matriz de conexión:** Todo grafo G tiene asociado una matriz M de dos dimensiones de orden $|N| \times |N|$, cuyos elementos representan la multiplicidad del par (n_i, n_j) .
Ejemplo: La matriz de conexión del digrafo D es



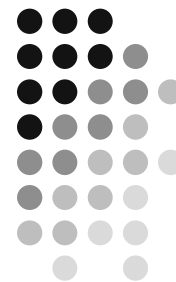
Conceptos básicos



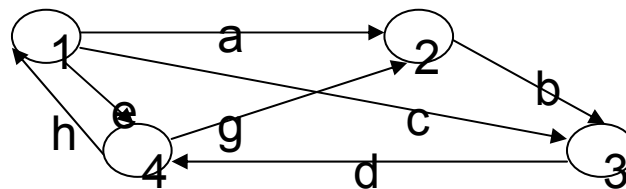
- **Camino:** Un camino C del digrafo D es una secuencia de arcos (a_1, a_2, \dots, a_n) tal que para todo par (a_i, a_j) de arcos consecutivos en C , el fin de a_i coincide con el inicio de a_j . Ejemplo: **b, d** es un camino, pero **a, c** no lo es.
- **Camino simple:** Es el camino que no recorre el mismo arco dos veces. Ejemplo: **b, d, g** es un camino simple, pero **b, d, g, b** no es un camino simple.
- **Camino elemental:** Es el camino que no visita un mismo nodo más de una vez. Ejemplo: **b, d** es un camino elemental, pero **b, d, g** no lo es.



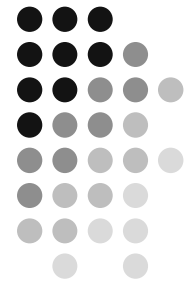
Conceptos básicos



- **Circuito:** Es un camino finito en el que el origen del primer arco coincide con el fin del último. Ejemplo: **a, b, d, h** es un circuito, pero **c, d, g** no lo es.
- **Circuito simple:** Un circuito es simple si a su vez es un camino simple. Ejemplo: **b, d, g** es un circuito simple, pero **e, h, e, h** es un circuito no simple.
- **Orden de un camino:** Es el número de arcos que conforman el camino. Ejemplo: Un bucle tiene un orden igual a 1. **b, d, g** es un circuito simple de orden 3.

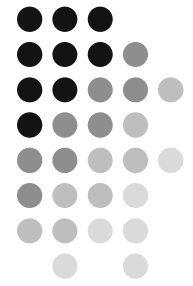


Conceptos básicos



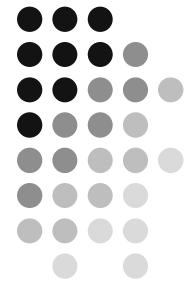
- **Nodo accesible:** Un nodo n_j es accesible desde otro nodo n_i , si
 - ❖ 1) $n_i = n_j$ o
 - ❖ 2) existe un camino desde n_i hasta n_j .
- **Nodo sucesor:** Un nodo n_j es sucesor de un nodo n_i si existe un camino que va desde n_i hasta n_j . Ejemplo: 2 es sucesor de 1.
- **Nodo predecesor:** Un nodo n_i es predecesor de otro nodo n_j si existe un camino que va desde n_i hasta n_j . Ejemplo: 1 es predecesor de 3.

Conceptos básicos

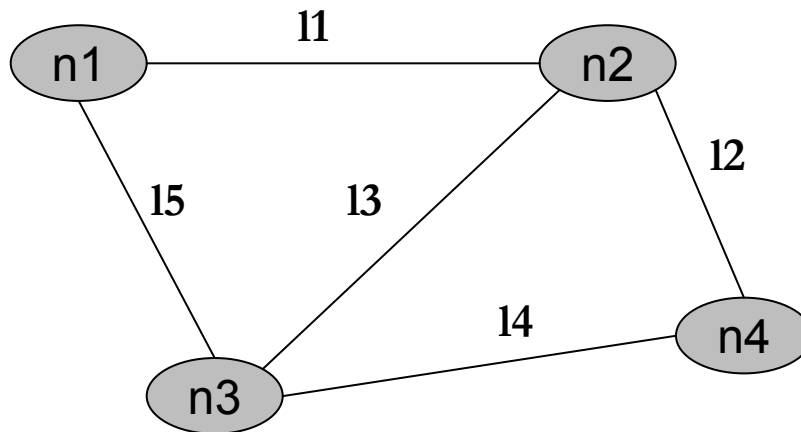


- **Grafo conexo:** Un grafo G es conexo si y sólo si sus nodos n_i no pueden ser particionados en dos conjuntos no vacíos N_1 y N_2 en los que sus caminos desde un punto cualquiera a otro estén en el mismo conjunto. Ejemplo: D es un grafo conexo.
- **Subgrafo:** Un subgrafo SG de G es un grafo constituido por un subconjunto SN de N y un subconjunto SA de A que conectan los nodos de SN . Ejemplo: $SG = \{ SN, SA, sf \}$ donde $SN \subset N$ y $SA \subset A$, $SN = \{2, 3, 4\}$, $SA = \{b, d, g\}$, $sf(b) = (2, 3)$, $sf(d) = (3, 4)$, $sf(g) = (4, 2)$.
- **Isomorfismo:** Dos grafos G y G' son isomórficos si existe una biyección $f: N \rightarrow N'$ tal que $(n_i, n_j) \in A \Leftrightarrow (f(n_i), f(n_j)) \in A'$.
- **Vecino:** En un dag G , el vecino de un nodo n es cualquier nodo adyacente a n en la versión no dirigida de G .

Conceptos básicos

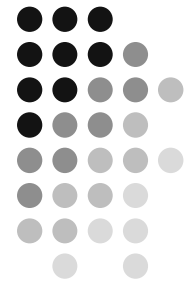


- Grafo no dirigido (GND): $G = \{N, L, f\}$



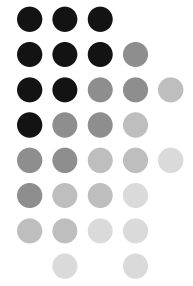
GND1

Conceptos básicos



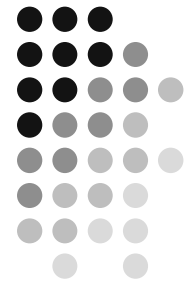
- **Cadena:** Para un grafo no dirigido GND es la secuencia de líneas (l_1, l_2, \dots, l_n) tal que el fin l_i coincide con el origen de l_{i+1} . Ejemplo: Para el grafo GND1, l_3, l_2, l_4 es una cadena, pero l_1, l_2 no lo es.
- **Cadena simple:** Es la cadena donde ninguna línea se repite. Ejemplo: Para el grafo GND1, l_3, l_2, l_4 es una cadena simple, pero l_3, l_2, l_4, l_4 no lo es.
- **Cadena elemental:** Es la cadena donde ningún nodo se repite. Ejemplo: Para el grafo GND1, l_3, l_2, l_4 es una cadena elemental, pero l_2, l_4, l_4 no lo es.

Conceptos básicos



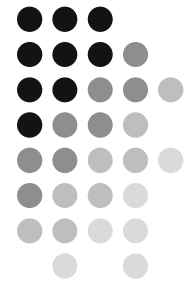
- **Ciclo:** Es una cadena finita donde el nodo inicial de la cadena coincide con el nodo terminal de la misma.
- **Ciclo simple:** Es el ciclo que a su vez es una cadena simple. Ejemplo: El grafo GND1 no tiene ciclos simples

Conceptos básicos



- **Grafo múltiple:** Es un grafo que contiene alguna arista paralela.
- **Digrafo acíclico:** Es un digrafo que no contiene circuitos. Se le conoce con las siglas **dag**.
- **Grafo o digrafo con peso:** Es un grafo o digrafo que tiene un valor entero o real asignado a cada arista.
- **Grafo completo:** Es un grafo no dirigido donde cada par de nodos es adyacente

Conceptos básicos



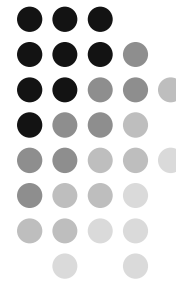
- **Grafo bipartito:** Es un grafo no dirigido que puede ser dividido en dos subgrafos sin conexión entre ellos.

$G=(N, A)$ donde $N_1 \in N$ y $N_2 \in N$ con $N_1 \cup N_2 = N$ y

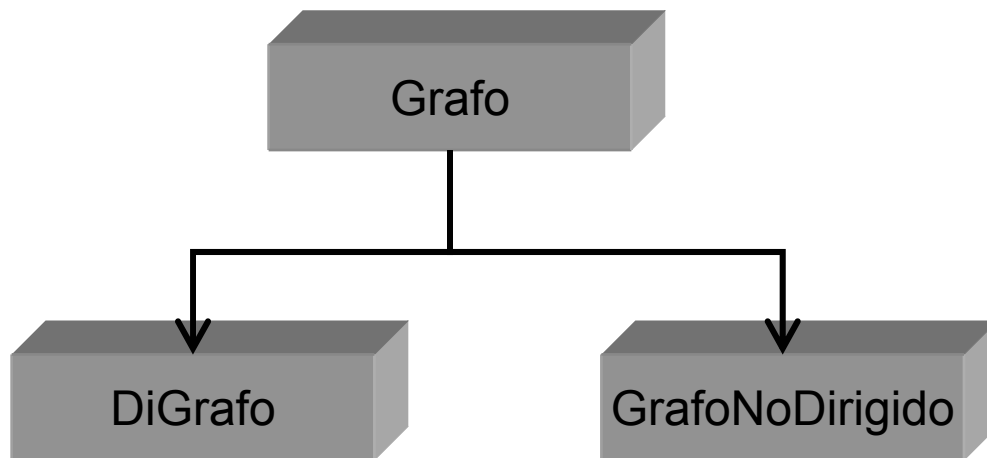
$\forall (i, j) \in A (i \in N_1 \wedge j \in N_2) \vee (i \in N_2 \wedge j \in N_1)$

- **Multigrafo:** Es un grafo no dirigido que puede tener varias aristas y lazos entre sus nodos.
- **Hipergrafo:** Es un grafo no dirigido con hiper-aristas que conectan varios nodos. Ejemplo: hipertexto e hipermedia.

Especificación del TAD Grafo

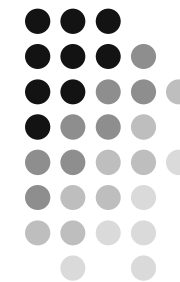


Un grafo se especifica en base a sus funciones básicas, entre las que se pueden mencionar: crearlo, insertar un nodo nuevo o una arista nueva, conocer si está vacío o no, consultar si un nodo existe, etc.



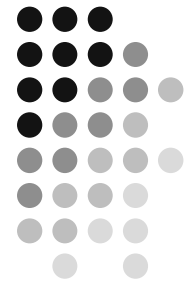
Grafos etiquetados
 $w: A \rightarrow \mathcal{R}$

Especificación del TAD Grafo



Marzo, 2005		Especificación Grafo[TipoEle]
1	<p>Sintáctica</p> <p>creaGrafo() → Grafo, nuevoNodo(Grafo, TipoEle) → Grafo, nuevaArista(Grafo, TipoEle, TipoEle) → Grafo, eliNodo(Grafo, TipoEle) → Grafo, eliArista(Grafo, TipoEle, TipoEle) → Grafo, conNodo(Grafo, TipoEle) → Lógico, conArista(Grafo, TipoEle, TipoEle) → Lógico, nodoAdyacente(Grafo, TipoEle) → Lista[TipoEle], vacíoGrafo(Grafo) → Lógico, destruyeGrafo() → .</p>	<p>-creaGrafo(): Crea un grafo vacío.</p> <p>-nuevoNodo(): Ingresa un nuevo nodo al grafo</p> <p>-nuevaArista(): Ingresa una nueva arista entre los dos nodos especificados, si existen.</p> <p>-eliNodo(): Elimina un nodo del grafo.</p> <p>-eliArista(): Elimina una arista del grafo entre los nodos especificados.</p> <p>-conNodo(): Regresa verdadero si existe en el grafo.</p> <p>-conArista(): Regresa verdadero si el arco existe.</p> <p>-nodoAdyacente(): Regresa la lista de los arcos adyacentes a él.</p> <p>-vacíoGrafo(): Regresa verdadero si es el grafo esta vacío</p> <p>-destruyeGrafo(): Destruye el grafo</p>
2	<p>Declaraciones</p> <p>TipoEle: no, {TipoNoDef}</p>	
3	<p>Semántica</p> <p>vacíoGrafo(creaGrafo()) = Verdadero vacíoGrafo(nuevoNodo(creaGrafo(), no)) = Falso conNodo(creaGrafo(), no) = Falso conNodo(nuevoNodo(creaGrafo(), no), no) = Verdadero conArista(creaGrafo(), no, no) = Falso conArista(nuevaArista(creaGrafo(), no, no), no, no) = Cierto eliNodo(creaGrafo(), no) = creaGrafo()</p>	

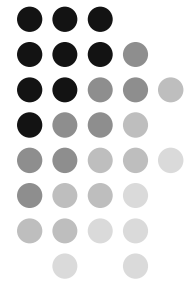
Implementación del TAD Grafo



- Para las estructuras no lineales de datos se tienen las mismas formas de almacenamiento que para las estructuras lineales, a saber: las que usan memoria continua, denominado aquí método secuencial y las que usan memoria dispersa o método enlazado
- Existen dos métodos para almacenar un grafo G :
 - ❖ Matriz de adyacencia: se implementa utilizando el método secuencial o con arreglos y
 - ❖ Listas de adyacencia implementada utilizando el método enlazado o listas enlazadas

Implementación de TAD

Grafo



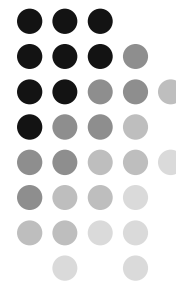
Marzo, 2005

Grafo[TipoEle]

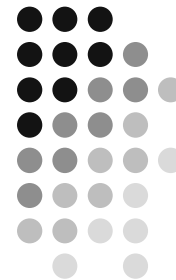
Clases: Arreglo de Entero, Entero, Tabla, TipoEleTab, Salida[TipoEle], TipoEle

<p>1 Superclase: ninguna</p> <p>2 Estructura: privado:</p> <p>3 Operaciones: público:</p> <p>Grafo()</p> <p>nuevoNodo(TipoEle: n)</p> <p>nuevaArista(TipoEle: ni, TipoEle: nf):</p> <p>eliNodo(TipoEle: n)</p> <p>eliArista(TipoEle: ni, TipoEle: nj)</p> <p>conNodo(TipoEle: n): Lógico</p> <p>conArista(TipoEle: ni, TipoEle: nf): Lógico</p> <p>nodoAdyacente(TipoEle: n, Entero: t): Arreglo[100] de TipoEle</p> <p>recorridoEnAmp()</p> <p>recorridoEnProf()</p> <p>rutaOptima(TipoEle: nj): Arreglo[100] de Salida[TipoEle]</p> <p>vacíoGrafo(): Lógico</p> <p>numNodos(): Entero</p> <p>numAristas(): Entero</p> <p>=(Grafo[TipoEle])</p> <p>~Grafo()</p>	<p>-Grafo(). <i>Constructor.</i> Crea un grafo vacío</p> <p>-nuevoNodo(). <i>Transformador.</i> Inserta un nuevo nodo al grafo.</p> <p>-nuevaArista(). <i>Transformador.</i> Ingresa un nuevo arco al grafo.</p> <p>-eliNodo(). <i>Transformador.</i> Elimina un nodo del grafo</p> <p>-eliArista(). <i>Transformador.</i> Elimina un arco del grafo.</p> <p>-conNodo(). <i>Observador.</i> Regresa verdadero, si existe.</p> <p>-conArista(). <i>Observador.</i> Regresa verdadero si existe el arco.</p> <p>-nodoAdyacente(). <i>Observador.</i> Regresa el arreglo de nodos adyacentes y su tamaño.</p> <p>-recorridoEnAmp(). <i>Observador.</i> Recorre el grafo en amplitud.</p> <p>-recorridoEnProf(). <i>Observador.</i> Recorre el grafo en profundidad</p> <p>-rutaOptima(). <i>Observador.</i> Regresa un arreglo con los nodos que conforman el camino de menos arcos a partir de un nodo inicial</p> <p>-vacíoGrafo(). <i>Observador.</i> Regresa verdadero si el grafo está vacío.</p> <p>-numNodos(), numAristas(). <i>Observadores.</i> Regresa el número de nodos y el de aristas del grafo, respectivamente.</p> <p>-=(). <i>Transformador.</i> Asigna un grafo.</p> <p>~Grafo(). <i>Destructor.</i> Destructor del grafo.</p>
---	--

Matriz de adyacencia



- Es un arreglo de dos dimensiones que representa las conexiones entre pares de nodos o vértices.
- Sea un grafo G con un conjunto de nodos N y un conjunto de aristas A . Suponga que el grafo es de orden n (número de nodos del grafo), donde $n > 0$.
- La matriz de adyacencia se representa por un arreglo de tamaño $n \times n$, donde:
$$M(i, j) = \begin{cases} 1 & \text{si existe un arco}(N_i, N_j) \text{ en } A, N_i \text{ es adyacente a } N_j \\ 0 & \text{en caso contrario} \end{cases}$$
- Las columnas y las filas de la matriz representan los nodos del grafo.
- Si existe una arista desde i hacia j (esto es, el nodo i es adyacente al nodo j) se almacena 1, si no existe la arista, se introduce 0
- Si el grafo es no dirigido, la matriz es simétrica $M(i, j) = M(j, i)$.



Matriz de adyacencia

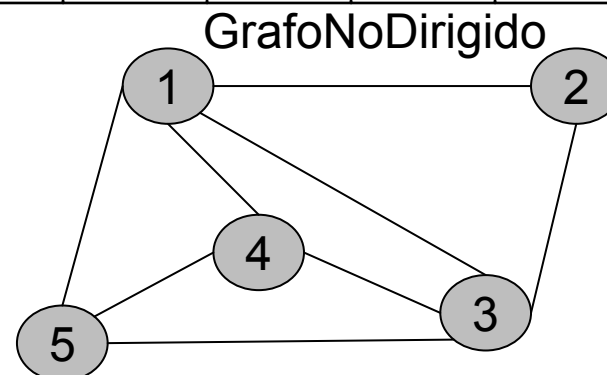
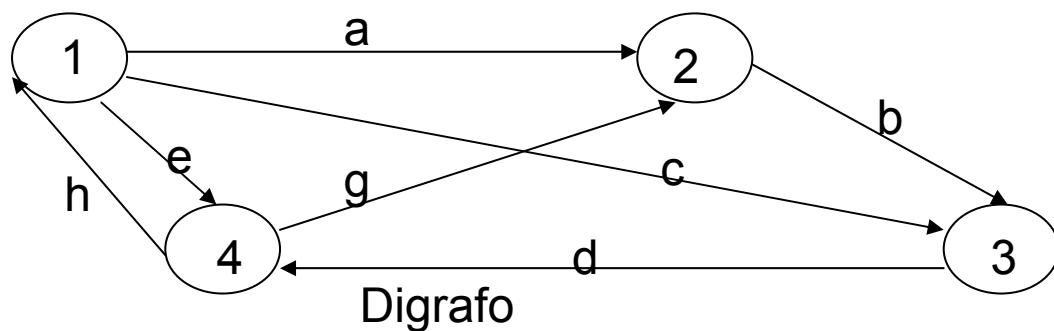
Al nodo

Del nodo	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	1
4	1	1	0	0

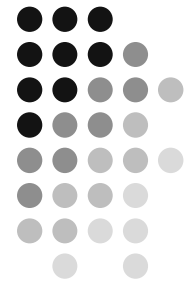
M del grafo no dirigido

	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	0	0
3	1	1	0	1	1
4	1	0	1	0	1
5	1	0	1	1	0

M del grafo dirigido

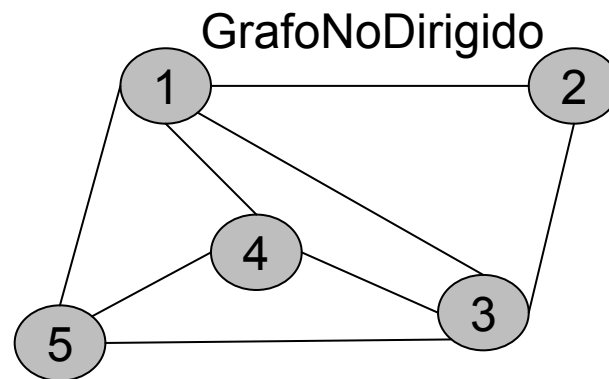
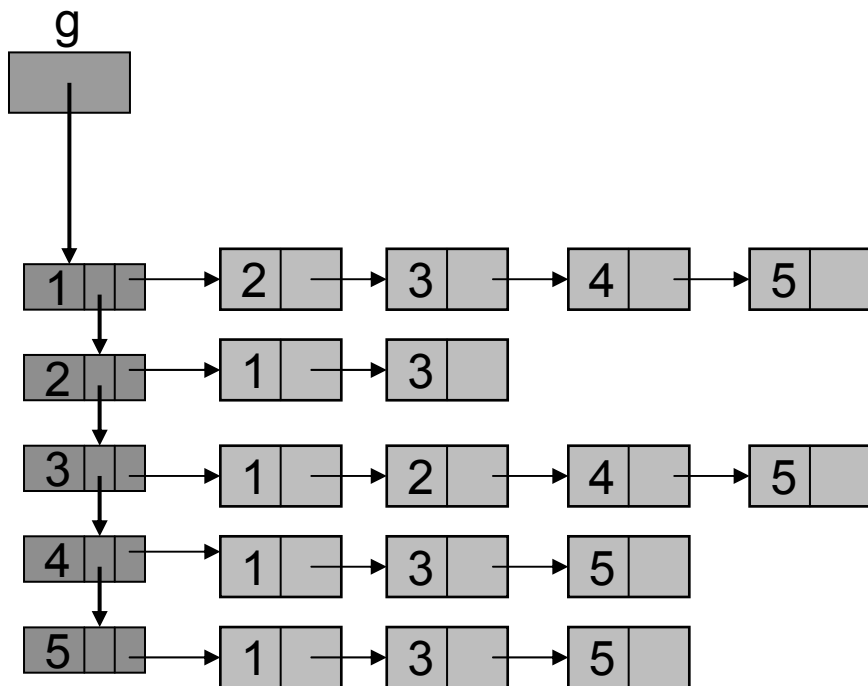
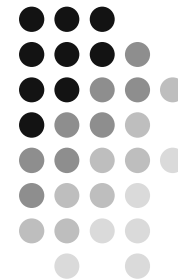


Listas de adyacencia

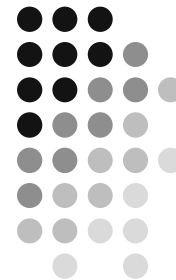


- El segundo método utilizado para representar grafos es útil cuando un grafo posee muchos nodos y pocas aristas (grafos no densos).
- Se utiliza una lista enlazada, llamada directorio, que almacena los nodos N del grafo y asociado a cada nodo del grafo se encuentra una lista enlazada con los nodos que son adyacentes a éste.
- Representa las aristas del grafo.
- Un grafo no dirigido de orden N con A aristas requiere N entradas en el directorio y $2 \cdot A$ entradas de listas enlazadas, excepto si existen bucles, que reduce el número de entradas a la lista en 1

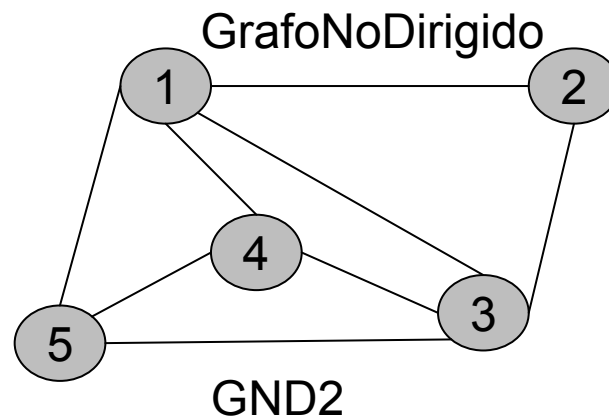
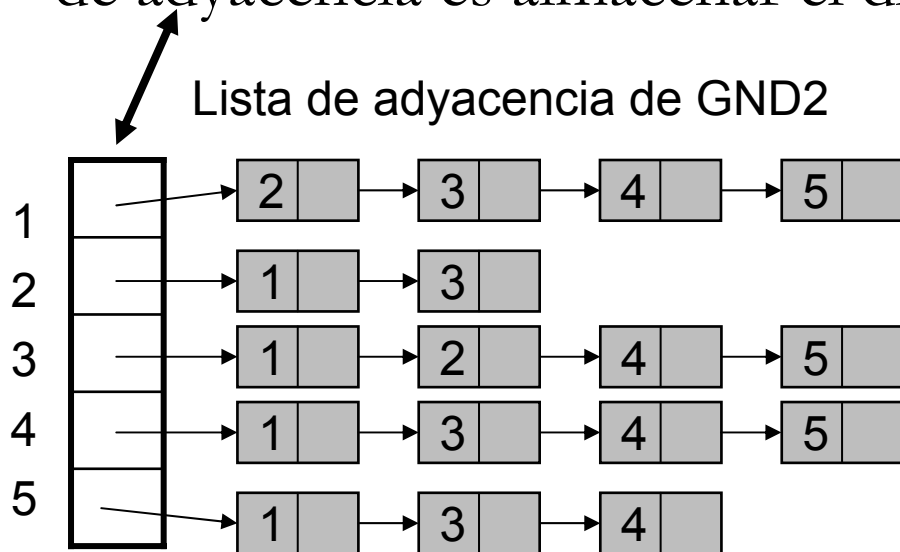
Listas de adyacencia



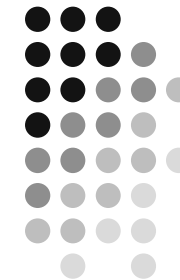
Listas de adyacencia



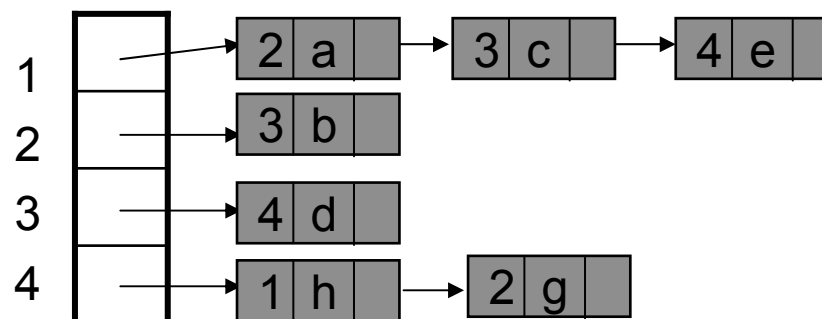
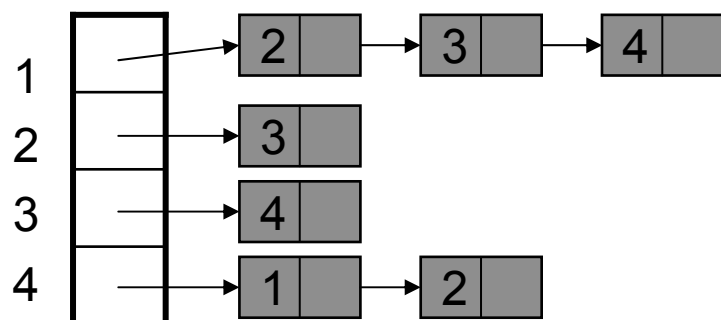
- Un grafo dirigido de orden N con A arcos requiere N entradas en el directorio y A entradas de listas enlazadas
- Una manera alterna de representar grafos utilizando listas de adyacencia es almacenar el directorio en un vector



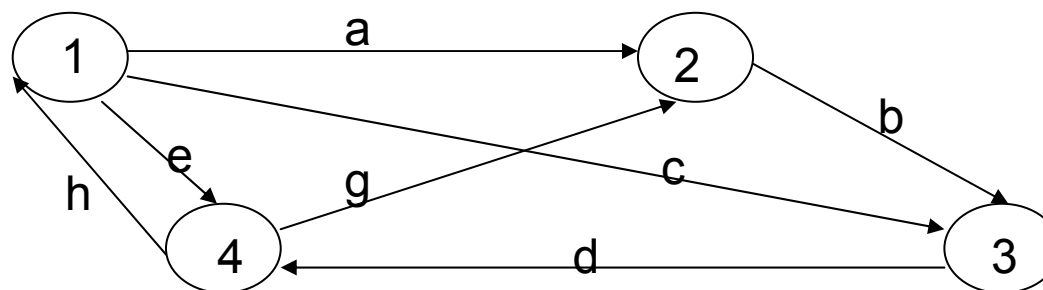
Listas de adyacencia



Lista de adyacencia del Digrafo1

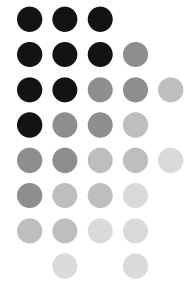


Lista de adyacencia del Digrafo1 etiquetado



Digrafo1

Implantación de un Digrafo usando matriz de adyacencia



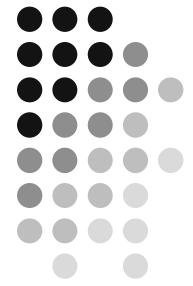
Marzo, 2005

DigrafoMat[TipoEle]

Clases: Arreglo de Entero+, Entero+, Tabla[TipoEleTab], Salida[TipoEle], TipoEle

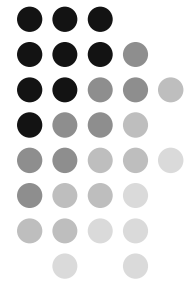
<p>1 Superclase: Grafo</p> <p>2 Estructura: privado: max: Entero+ = 100 pos: Entero+ = 0 ar: Entero+ = 0 g: Arreglo[100,100]De Entero + dir: Arreglo[100]De TipoEle t: Tabla[TipoEleTab]</p> <p>3 Operaciones: público: DigrafoMat() nuevoNodo(TipoEle: n) nuevoArco(TipoEle: ni, TipoEle: nf) eliNodo(TipoEle: n) eliArco(TipoEle: ni, TipoEle: nj) conNodo(TipoEle: n): Lógico conArco(TipoEle: ni, TipoEle: nf): Lógico nodoAdyacente(TipoEle: n, Entero+: t): ArregloDeTipoEle recorridoEnAmp() recorridoEnProf() rutaOptima(TipoEle: nj): ArregloDe Salida[TipoEle] numNodos(): Entero numAristas(): Entero incidenciaEnt(TipoEle: n): Entero incidenciaSal(TipoEle: n): Entero vacíoDigrafo(): Lógico =(DigrafoMat[TipoEle]) ~DigrafoMat()</p>	<p>-max: maximo número de elementos del arreglo.</p> <p>-pos: Próximo nodo libre en la matriz.</p> <p>-ar: Número actual de arcos en el digrafo</p> <p>-g: Matriz de adyacencia para representar el digrafo.</p> <p>-dir: Almacena los nombres de cada nodo en la posición correspondiente en la matriz.</p> <p>-t: Permite guardar el nombre del nodo con su posición en la matriz.</p> <p>-DigrafoMat(). Constructor. Crea un digrafo vacío</p> <p>-nuevoNodo(). Transformador. Inserta un nuevo nodo en el digrafo.</p> <p>-nuevoArco(). Transformador. Ingresa un nuevo arco al digrafo.</p> <p>-eliNodo(). Transformador. Elimina un nodo del digrafo</p> <p>-eliArco(). Transformador. Elimina un arco del digrafo.</p> <p>-conNodo(). Observador Regresa el nodo si existe.</p> <p>-conArco(). Observador. Regresa verdadero si existe el arco.</p> <p>-nodoAdyacente(). Observador. Regresa el arreglo de nodos adyacentes y su tamaño t.</p> <p>-recorridoEnAmp(). Observador. Recorrido en amplitud</p> <p>-recorridoEnProf(). Observador. Recorrido en profundidad</p> <p>-rutaOptima(). Observado. Regresa en un arreglo los nodos que conforman el camino con menos arcos.</p> <p>-numNodos(), numAristas(). Observadores. Regresa el número de nodos y de aristas del grafo, respectivamente.</p> <p>-incidenciaEnt(). Observador. Da el grado de incidencia de entrada de un nodo.</p> <p>-incidenciaSal(). Observador. Da el grado de incidencia de salida de un nodo.</p> <p>-vacíoDigrafo(). Observador. Regresa verdadero si el digrafo está vacío.</p> <p>-=(.). Transformador. Asignación, regresa un digrafo igual en valores al enviado por parámetro.</p> <p>~DigrafoMat(). Destructor.</p>
---	--

Implementación de la clase Salida



Marzo/05		
Salida[TipoEle] Clases: Entero+, TipoEle		
1	Superclase: ninguna	
2	Estructura: privado: dis: Entero+ =0 padre: TipoEle	-dis: Número de arcos recorridos. -padre: Nombre del nodo padre adyacente a él. -Dis(). <i>Observador y transformador.</i> Almacena la distancia del nodo origen a ese nodo.
3	Operaciones: público: Salida() Dis(): Entero Dis(Entero: d) Padre(): TipoEle Padre(TipoEle: el)	-Padre(): <i>Observador y transformador.</i> Almacena el valor del nodo que lo antecede en el camino desde el nodo origen.

Implantación de un Digrafo usando listas de adyacencia



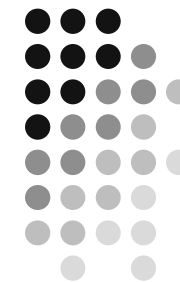
Marzo, 2005

DigrafoLis[TipoEle]

Clases: TipoEle, Nodo1[TipoEle], Lista[Nodo1[TipoEle]], ArregloDe[TipoEle]

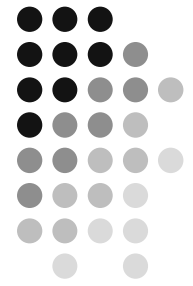
<p>1 Superclase: Grafo</p> <p>2 Estructura: privado: g: Lista[Nodo1[TipoEle]]</p> <p>3 Operaciones: público: DigrafoLis() nuevoNodo(TipoEle: n) nuevoArco(TipoEle: ni, TipoEle: nf) eliNodo(TipoEle: n) eliArco(TipoEle: ni, TipoEle: nj) conNodo(TipoEle: n): Lógico conArco(TipoEle: ni, TipoEle: nf): Lógico nodoAdyacente(TipoEle: n, Entero+: t): Lista[TipoEle] recorridoEnAmp() recorridoEnProf() rutaOptima(TipoEle: n)Lista[Salida[TipoEle]] numNodos(): Entero numAristas(): Entero incidenciaEnt(TipoEle: n): Entero incidenciaSal(TipoEle: n): Entero vacíoDigrafo(): Lógico =(DigrafoLis[TipoEle]) ~DigrafoLis()</p>	<p>-g: Lista de adyacencia para representar el DigrafoLis.</p> <p>-DigrafoLis(). <i>Constructor</i>. Crea un digrafo vacío</p> <p>-nuevoNodo(). <i>Transformador</i>. Inserta un nuevo nodo en el digrafo.</p> <p>-nuevoArco(). <i>Transformador</i>. Ingresa un nuevo arco al digrafo.</p> <p>-eliNodo(). <i>Transformador</i>. Elimina un nodo del digrafo</p> <p>-eliArco(). <i>Transformador</i>. Elimina un arco del digrafo.</p> <p>-conNodo(). <i>Observador</i>. Regresa verdadero si el nodo existe. Deja el cursor de la lista antes del nodo que lo contiene.</p> <p>-conArco(). <i>Observador</i>. Regresa verdadero si existe el arco.</p> <p>-nodoAdyacente(). <i>Observador</i>. Regresa el arreglo de nodos adyacentes y su tamaño t.</p> <p>-recorridoEnAmp(). <i>Observador</i>. Recorre el grafo en amplitud.</p> <p>-recorridoEnProf(). <i>Observador</i>. Recorre el grafo en profundidad.</p> <p>-rutaOptima(). <i>Observador</i>. Regresa en un arreglo los nodos que conforman el camino con menos arcos.</p> <p>-incidenciaEnt(). <i>Observador</i>. Da el grado de incidencia de entrada de un nodo.</p> <p>-incidenciaSal(). <i>Observador</i>. Da el grado de incidencia de salida de un nodo.</p> <p>-vacíoDigrafo(). <i>Observador</i>. Regresa verdadero si el digrafo está vacío.</p> <p>numNodos(), numAristas(). <i>Observadores</i>. Regresa el número de nodos y de aristas del grafo, respectivamente.</p> <p>-=(). <i>Transformador</i>. Asignación. Regresa un digrafo igual en valor al pasado por parámetro</p> <p>-~DigrafoLis(). <i>Destructor</i>.</p>
--	--

Implantación de la clase Nodo1



Marzo, 2005		Nodo1[TipoEle] Clases: Entero, TipoEle, ApuntadorA, Lista[TipoEle]	
1 2	Superclases: ninguna Estructura: privado : infor: TipoEle padre: TipoEle distancia: Entero+ listaAdya: Lista[TipoEle] pSig: ApuntadorA Nodo1[TipoEle] = Nulo	-infor. Nombre del nodo. -padre. Nombre del nodo padre. -distancia. Distancia para llegar a ese nodo desde el nodo inicio. -listaAdy. Lista de adyacencia de cada nodo. -pSig. Apuntador al siguiente nodo. -Nodo1(): <i>Constructor</i> -Infor(). <i>Observador y transformador.</i> Da la identificación del nodo. -Padre(). <i>Observador y transformador.</i> Da la información del nodo padre en el calculo del camino. -Distancia(). <i>Observador y transformador.</i> Da la distancia al nodo en el cálculo del camino. -ListaAdya(). <i>Observador y transformador.</i> Lista de nodos adyacentes. -PSig(): <i>Observador y transformador.</i> Apuntador al siguiente nodo.	
3	Operaciones: público : Nodo1() Infor(): TipoEle Infor(TipoEle: e) ListaAdya(): Lista[TipoEle] ListaAdya(Lista[TipoEle]: lad) Padre(): TipoEle Padre(TipoEle: p) Distancia(): Entero Distancia(Entero: d) PSig(): ApuntadorA Nodo1[TipoEle] PSig(ApuntadorA Nodo1[TipoEle]: pn)		

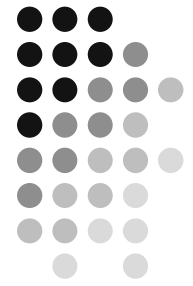
Implementación de las clases Nodo y Salida



Marzo/05		
Nodo[TipoEle] Clases: TipoEle, ApuntadorA		
1	Superclases: ninguna	-info. Información del nodo adyacente. -pSig. Apuntador al siguiente nodo adyacente. -Nodo(). <i>Constructor.</i> -Info(). <i>Observador y transformador.</i> Da la información del nodo adyacente. -PSig(). <i>Observador y transformador.</i> Da la información del apuntador al siguiente nodo adyacente.
2	Estructura: privado: info: TipoEle pSig: ApuntadorA Nodo[TipoEle] = Nulo	
3	Operaciones: público: Nodo() Info(): TipoEle Info(TipoEle: e) PSig():ApuntadorA Nodo[TipoEle] PSig(ApuntadorA Nodo[TipoEle]: pn)	

Marzo/05		
Salida[TipoEle] Clases: Entero, TipoEle		
1	Superclase: ninguna	-nNodo: Nombre del nodo. -padre: Nombre de su nodo padre adyacente. -NNodo(). <i>Observador y transformador.</i> Almacena la distancia del nodo origen a ese nodo. -Padre(). <i>Observador y transformador.</i> Almacena el valor del nodo que lo antecede en el camino desde el nodo origen.
2	Estructura: privado: nNodo: TipoEle padre: TipoEle	
3	Operaciones: público: Salida() NNodo(): TipoEle NNodo(TipoEle: n) Padre(): TipoEle Padre(TipoEle: el)	

Implementación de la clase Nodo2



Nodo2[TipoEle] Clases: TipoEle, ApuntadorA		
1	Superclases: ninguna	<p>-pAnt: Dirección del nodo anterior.</p> <p>-info: Información del nodo.</p> <p>-pSig: Dirección del nodo siguiente.</p> <p>-Nodo2(). Constructor. Crea un nodo con un elemento no definido y dos apuntadores nulos.</p> <p>-PAnt(). Transformador. Modifica el apuntador al nodo anterior por la dirección que viene en el parámetro.</p> <p>-Info(). Transformador. Modifica la información del nodo por la que viene en el parámetro.</p> <p>-PSig(). Transformador. Modifica el apuntador al nodo siguiente por la dirección que viene en el parámetro.</p> <p>-PAnt(). Observador. Regresa la dirección del nodo anterior.</p> <p>-Info(). Observador. Devuelve la información contenida en el nodo.</p> <p>-PSig(). Observador. Regresa la dirección del nodo siguiente.</p>
2	Estructura: privado: pAnt : ApuntadorA Nodo2[TipoEle] = Nulo info: TipoEle = {TipoEleNoDef} pSig : ApuntadorA Nodo2[TipoEle] = Nulo	
3	Operaciones: público: Nodo2() PAnt(ApuntadorA Nodo2[TipoEle]: pa) PAnt() : ApuntadorA Nodo2[TipoEle] Info(TipoEle: e) Info(): TipoEle PSig(ApuntadorA Nodo2[TipoEle]: pn) PSig(): ApuntadorA Nodo2[TipoEle]	