****

**RESUMEN GRAFOS:**

***DEFINICION GRAFO DIRIGIDO O DIGRAFO: (Los arcos poseen dirección)***

* Un grafo G = (V, E) es un conjunto V vértices (o nodos) y un conjunto de aristas dirigidas (o arcos dirigidos)
* Intuitivamente E permite representar una relación entre elementos de V

Diagrama

Descripción generada automáticamente con confianza media



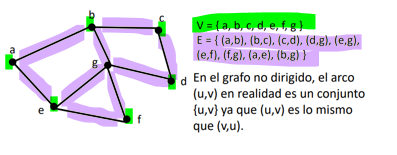
Los arcos poseen dirección, por lo tanto los arcos dirigidos

**ARCOS DIRIGIDOS**

**NODOS O VERTICES**

***DEFINICION DE GRAFO NO DIRIGIDO: (Los arcos no poseen dirección)***

* Un grafo G = (V, E) es un conjunto V de vértices (o nodos) y un conjunto ***E ⊆ V x V*** de aristas (o arcos)
* Intuitivamente E permite representar una relación (simétrica) entre elementos de V



Los arcos no poseen dirección, por lo tanto los arcos dirigidos

**ARCOS SIN DIRECCION**

**NODOS O VERTICES**

Extiende a



Diagrama

Descripción generada automáticamente con confianza media

Diagrama

Descripción generada automáticamente

**Otros ejemplos:**

1. Grafo de colaboración entre autores y coautores científicos
2. Grafo que representa la colección de documentos de HTML enlazados por hipervínculos (un documento a.html contiene un hipervínculo al documento b.html) (cada vértice es un documento con una URL y los hipervínculos dentro de un documento son los arcos dirigidos)

**ARCOS EMERGENTES:** Arcos que emergen directo del vértice a tratar

**GRADO DE EMERGENCIA:** Cantidad de arcos emergentes de un vértice

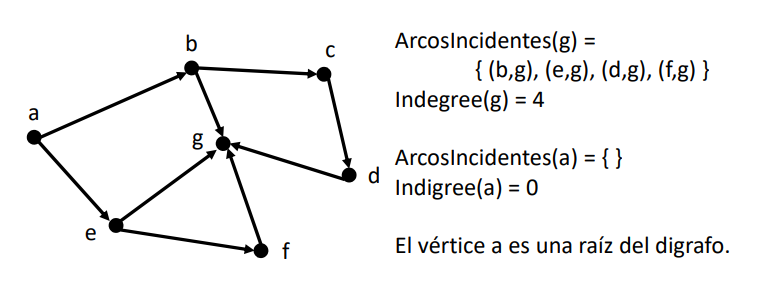
**HOJA DEL DIGRAFO:** Son aquellos vértices que no poseen arcos emergentes, o grado de emergencia igual a 0

Gráfico, Gráfico radial

Descripción generada automáticamente

**ARCOS INCIDENTES:** Arcos que llegan a un vértice determinado

**GRADO DE INCIDENCIA:** Cantidad de arcos incidentes que llegan a un vértice dado

**RAICES DEL DIGRAGO:** Son aquellos vértices que no poseen arcos incidentes, o grado de incidencia igual a 0

**ALGUNAS ACLARACIONES PARA GRAFOS NO DIRIGIDOS:**

* En un grafo no dirigido, se habla de grado porque los arcos que inciden y los que emergen son los mismos
* Se hablará solo de arcos incidentes (Arcos que inciden igual que grados que emergen)

Gráfico, Gráfico radial

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente**GRAFO PESADO:** Grafo donde las aristas contienen una etiqueta (o peso cuando las etiquetas son numéricas)



**Ejemplo:** Dígrafo donde los vértices son aeropuertos y los arcos representan los vuelos (distancia entre aeropuertos, si los pesos son numéricos)

**MULTI(DI)GRAFO:** (Di)grafo donde un par de vértices puede estar conectado por una colección de arcos. Es decir, puede haber más de un arco entre cada par de vértices (se los denomina **ARCOS PARALELOS**)

**Gráfico, Gráfico radial

Descripción generada automáticamente**

**Diagrama

Descripción generada automáticamenteCAMINO:** Un camino de V a W es una secuencia alternante de vértices y arcos tales que empiezan en V y termina en W, y cada arco es incidente en sus vértices predecesor y sucesor

Imagen que contiene Diagrama

Descripción generada automáticamente

**CICLO (SIMPLE):** Un camino que comienza y termina en el mismo vértice (Sin repetir arcos) 🡪 Los arcos

**CICLO (CAMINO) DIRIGIDO:** Un ciclo (camino) donde las aristas tienen dirección y son recorridas en su dirección

**ACICLICO (ARBOL):** Es un grafo que no posee ciclos

Gráfico, Gráfico radial

Descripción generada automáticamente

**Gráfico, Gráfico radial

Descripción generada automáticamenteCOSTO DE UN CAMINO (CICLO):** Suma de los pesos de los arcos que forman el camino (ciclo)

**Gráfico, Gráfico radial

Descripción generada automáticamenteSUBGRAFO:** Un subgrafo H de un grafo G es un grafo donde los vértices de H son un subconjunto de los vértices de G y los arcos de H son un subconjunto de los arcos de G.

**GRAFO CONEXO:** Un grafo G es conexo si para dos vértices cualquiera de G hay un camino entre ellos

**GRAFO NO CONEXO:** Lo contrario a grafo conexo; existe al menos dos vértices del grafo que no existe un camino entre ellos 🡪 Si un grafo no es no es conexo, sus **SUBGRAFOS MAXIMALES CONEXOS** se llaman **COMPONENTES CONEXAS**

**GRAFO FUERTEMENTE CONEXO:** Grafos donde cada par de vértices (cualquiera) su camino es dirigido [Siempre se da en grafos dirigidos]

Gráfico, Gráfico radial

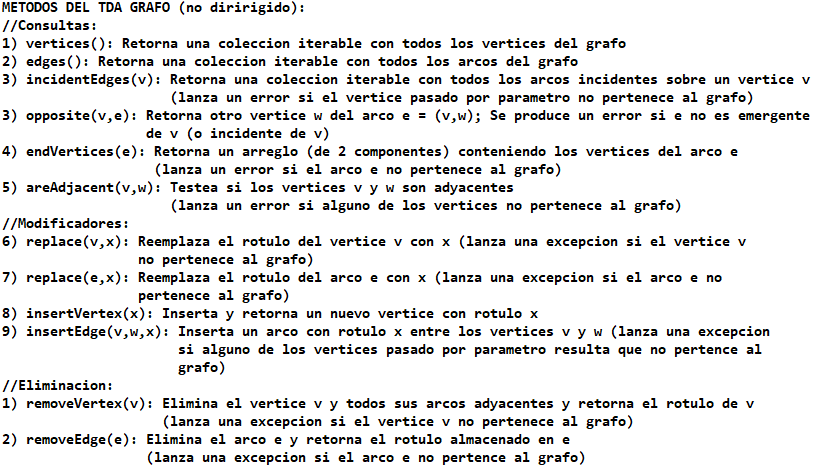
Descripción generada automáticamente

**TDA GRAFO: (Grafo no dirigido)**

El tipo de dato abstracto Grafo exporta tres sorts [Interfaces]:

1. Graph<V, E> 🡪 Un grafo pesado de vértices con rótulos del tipo V y arcos con rotulo del tipo E
2. Vertex<V, E> 🡪 La posición de un vértice con rotulo del tipo V [Vértices]
3. Edge<V, E> 🡪 La posición de un arco con rotulo de tipo E [Arcos]

Las dos últimas son equivalentes a la interfaz Position<E> 🡪 Posiciones de arcos y vértices en un grafo



Texto, Carta

Descripción generada automáticamente

**ACLARACIONES:**

1. Si los arcos o vértices no poseen rotulo, me devuelve null
2. Va 🡪 Vértice a
3. 🡪 Arco (a,b)
4. Determina si ambos vértices están conectados por un arco

**(3)**

Gráfico, Gráfico radial

Descripción generada automáticamente

**(2)**

**(1)**

**(4)**

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

**ESTRUCTURAS DE DATOS PARA LA IMPLEMENTACION DE GRAFOS (SIMPLES Y PESADOS):**

1. **LISTA DE ARCOS:**

* El grafo es una lista de vertices y una lista de arcos (los vertices y los arcos conocen sus rotulos respectivos)
* Los vertices conocen sus rotulos respectivos y su posicion en la lista de vertices (Permite eliminar en O(1), si no hay arcos involucrados)
* Los arcos conocen su peso, los vertices que unen y su posicion en la lista de arcos (Permite eliminar arcos en orden O(1))

Diagrama, Texto

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

* **ORDENES DE EJECUCION DE LOS METODOS DE LA IMPLEMENTACION:** Dado un grafo G= (V,A) , sean:
* **N =** la cantidad de vertices del grafo
* **M =** la cantidad de arcos del grafo

**Aclaraciones:**

1. Depende de la cantidad de vértices
2. Depende de la cantidad de arcos
3. Depende de la cantidad de arcos (posición de arcos en la lista)
4. Depende de la cantidad de arcos adyacentes al vértice (posición o existencia en la lista de arcos)

|  |  |
| --- | --- |
| **OPERACION** | **TIEMPO (ORDEN)**  **(1)** |
| **vertices()** | **O(n)**  **(2)** |
| **edges()** | **O(m)** |
| **endVertices(e)** | **O(1)** |
| **opposite(v,e)** | **O(1)**  **(3)** |
| **incidentEdges(v)** | **O(m)**  **(3)** |
| **areAdjacent(v,w)** | **O(m)** |
| **replace(v,x)** | **O(1)** |
| **replace(e,x)** | **O(1)** |
| **inserVertex(x)** | **O(1)** |
| **insertEdges(v,w,x)** | **O(1)** |
| **removeEdge(e)** | **O(1)**  **(4)** |
| **removeVertex(v)** | **O(m)** |

**//La implementacion mas ineficiente**

1. **LISTA DE ADYACENCIAS:**

* El grafo conoce una lista de vertices y una lista de arcos
* Cada vertice conoce su rotulo y los arcos que inciden en el
* Los arcos conocen los vertices que unen y su peso
* La mejora que propone la lista de adyacencia es introducer una lista que para cada vertice indica los arcos que emergen cuando el grafo es dirigido y los que emergen/inciden cuando el grafo es no dirigido

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

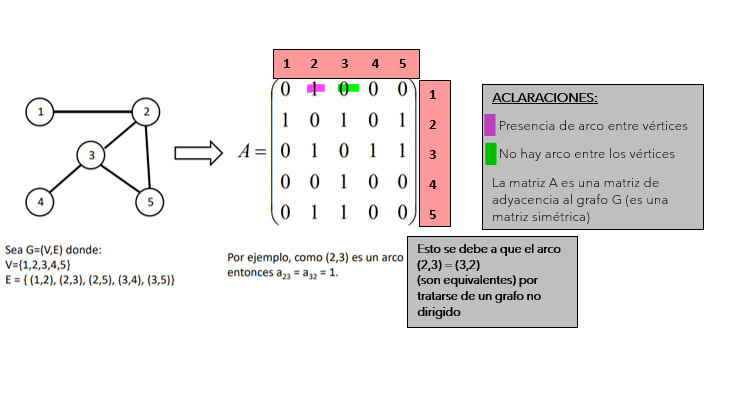
* **ORDENES DE EJECUCION DE LOS METODOS DE LA IMPLEMENTACION:** Dado un grafo G= (V,A) , sean:
* **N** = la cantidad de vertices del grafo
* **M** = la cantidad de arcos del grafo

Tabla

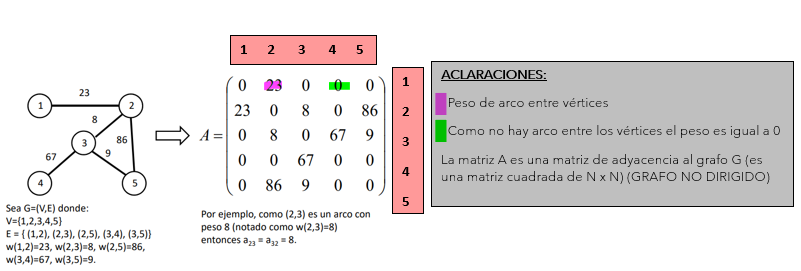
Descripción generada automáticamente

**ENFOQUES MATEMATICOS PARA REPRESENTAR UN GRAFO CON UNA MATRIZ DE ADYACENCIAS (MATRIZ SIMETRICA):**

* Un **grafo G** no dirigido con vertices numericos se pueden representar matematicamente por una **matriz booleana** donde quiere decir que hay un arco entre los vertices “I” y “j”**,** yquiere decir que no hay un arco entre “I” y “j”



* Un **grafo G no dirigido pesado** con vertices numericos se puede representar matematicamente por **una matriz real**  donde es el peso del arco

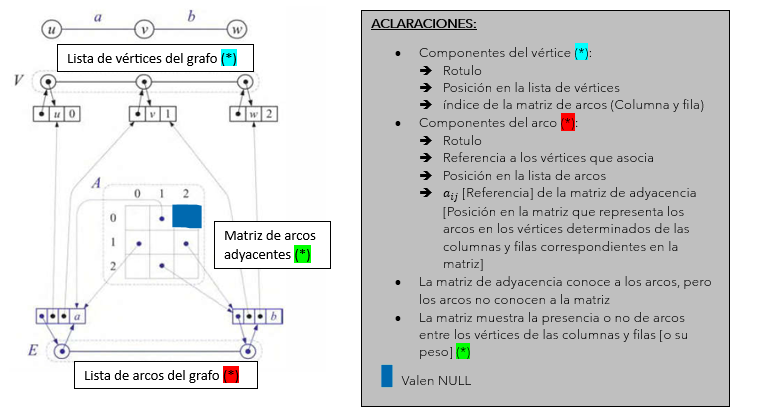


1. **MATRIZ DE ADYACENCIAS:**

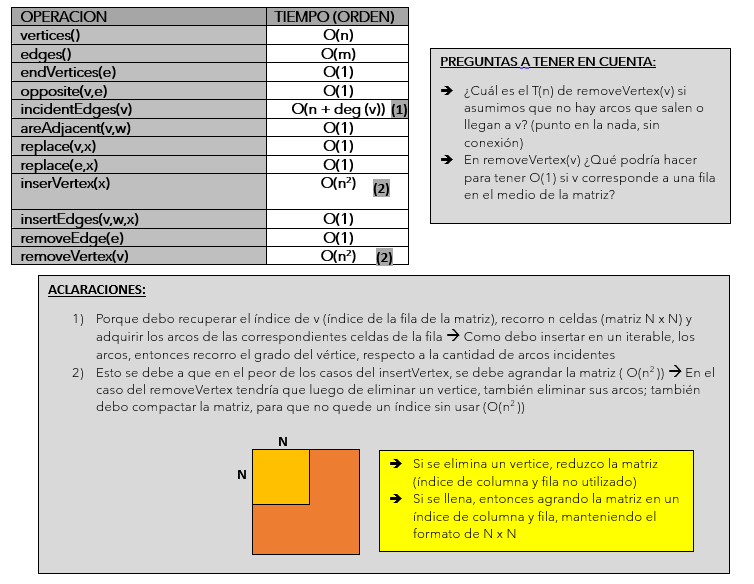
**VENTAJA DE LA IMPLEMENTACION:** Fácil de programar

**DESVENTAJAS DE LA IMPLEMENTACION:**

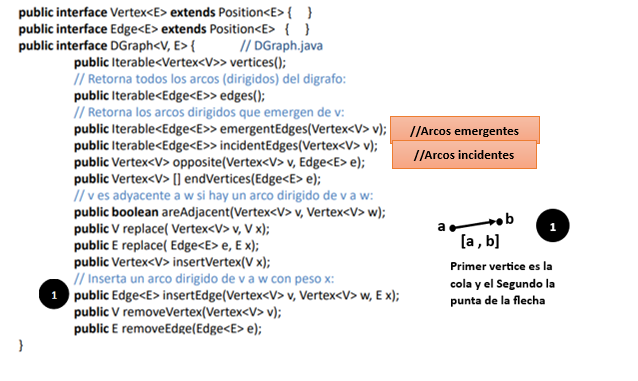
* Fragmentación interna en la estructura de datos
* Desperdicio de memoria
* En la representacion de **matriz de adyacencias** ademas de **la matriz de arcos** tendremos una **lista de vertices y una lista de arcos**
* Cada **vértice** conoce su posición en la lista de vértices, el rótulo del vértice y su índice en la matriz de arcos **(si tenemos n vértices, los índices de vértices van de 0 a n-1)**
* Cada **arco** conoce su posición en la lista de arcos, los vértices que conecta y el peso del arco



* **ORDENES DE EJECUCION DE LOS METODOS DE LA IMPLEMENTACION:** Dado un grafo G= (V,A) , sean:
* **N =** la cantidad de vertices del grafo
* **M =** la cantidad de arcos del grafo



**TDA GRAFO: (Grafo dirigido)**

* Los arcos tienen direccion entonces
* Las operaciones de **insertEdge(v,w,x)** inserta un arco dirigido de v a w con peso x
* Agregamos una nueva operacion: **emergentEdges(v)** que retorna un iterable con los arcos que emergen de v
* El almacenamiento de los arcos que inciden en un vertice puede o no mantenerse dependiendo de si nos interesa soportar la operacion **incidentEdges(v)**

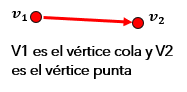
IMPLEMENTACION DE VERTICE, ARCO E INTERFAZ DEL DIGRAFO (TDA)

* **IMPLEMENTACION DEL DIGRAFO CON LISTA DE ADYACENCIAS:**

Diagrama

Descripción generada automáticamente

DIAGRAMA UML DE IMPLEMENTACION DEL DIGRAFO CON LISTA DE ADYACENCIAS

* **IMPLEMENTACION DEL DIGRAFO CON MATRIZ DE ADYACENCIAS (MODIFICACIONES):**
* **Necesitamos una lista de vertices y una lista de arcos (dirigidos)**
* La representacion de **los vertices** es igual (a la de lista de adyacencias)
* La representacion de **los arcos** asume que un vertice es la cola del arco y el otro es la punta **(1)**
* **insertVertex(v,w,x)** inserta un arco dirigido de v a w

**(1)**

* **insidentEdges (v)** debe recorrer la columna de la matriz correspondiente al indice de v (O(n)) (Tener en cuenta **(1)**)

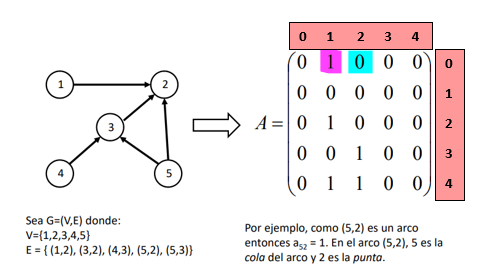
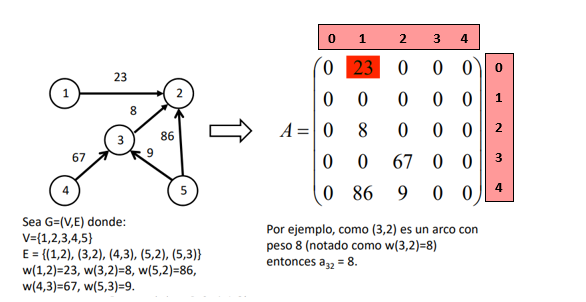
**//Arcos emergentes** 🡪 Son las filas de la matriz (segun el indice del vertice)

**//Arcos incidents** 🡪 Son las columnas de la matriz (segun el indice del vertice)

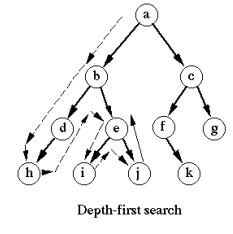
* **emergentEdges(v)** debe recorrer la fila de la matriz correspondiente al indice de v (O(n))
* **endVertices(e)** retorna un arreglo “a” donde a[0]=cola(e) y a[1]=punta(e) (, tiene orden)

//Agregar de ser posible los tiempos de ejecucion de los metodos de un grafo implementado con matriz de adyacencia

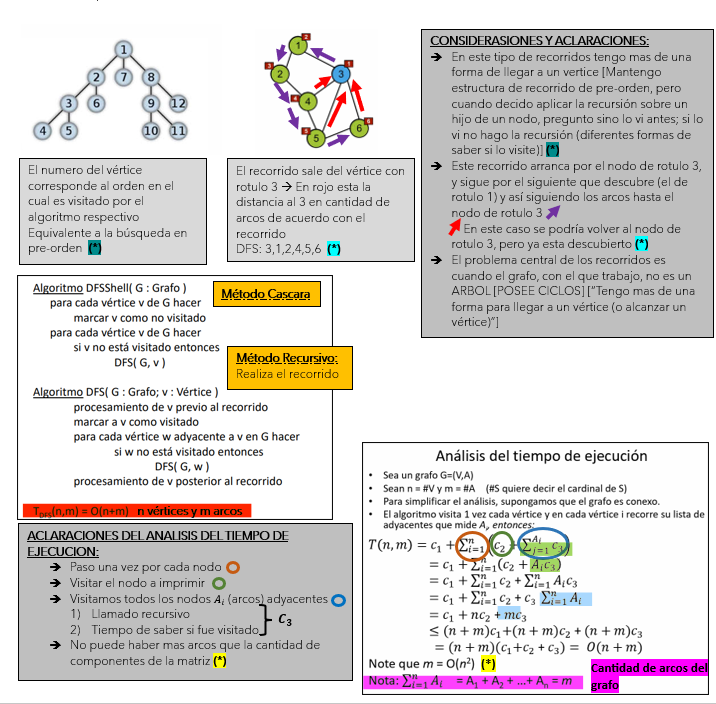
**ENFOQUES MATEMATICOS PARA REPRESENTAR UN GRAFO DIRIGIDO CON UNA MATRIZ DE ADYACENCIAS:**

1. **MATRIZ DE UN GRAFO DIRIGIDO (SIN PESO EN LOS ARCOS):** Un grafo G dirigido con vértices numéricos se puede representar matemáticamente por una matriz booleana donde quiere decir que hay un arco dirigido entre los vértices i y j y quiere decir que no hay un arco dirigido entre i y j
2. **MATRIZ DE UN GRAFO DIRIGIDO PESADO:** Un grafo G dirigido pesado con vértices numéricos se puede representar matemáticamente por una matriz real donde es el peso del arco dirigido (i,j)

**RECORRIDOS DE GRAFOS:**

1. **RECORRIDO EN PROFUNDIDAD (DEPTH-FIRST SEARCH o DFS):**

* Equivale al recorrido pre o post orden en árboles + un testeo para no volver a recorrer un subgrafo ya explorado : a, b, d, h, e, i, j, c, f, k, g **(ver imagen)**
* Una búsqueda en profundidad (DFS o Depth-First Search) permite recorrer todos los vértices de un grafo de manera ordenada
* Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto
* Cuando ya no quedan más nodos que visitar en dicho camino, regresa (backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado [El GRAFO PUEDE SER NO CONEXO, en tal caso nose por cual vertice arrancar]



**OPCIONES DE IMPLEMENTACION DE LAS MARCAS DE NODOS VISITADOS:**

1. Usar un mapeo externo al grafo de Vertex<V> en boolean

* VENTAJA: No hay que modificar el grafo que ya programamos
* DESVENTAJA: El tiempo de marcar y desmarcar puede tener O(n)

1. Agregar un boolean a la clase vertice del grafo

* VENTAJA: Puede garantizar que marcar y desmarcar funciona en O(1)
* DESVENTAJA: Por cada algoritmo que escribo tengo que ensuciar el grafo agregando atributos y operaciones

1. Decorar los vértices del grafo (opción del libro GT)

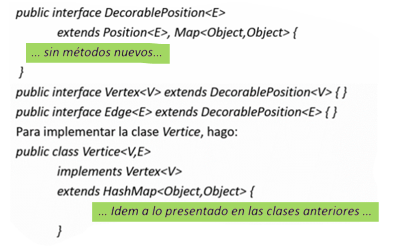
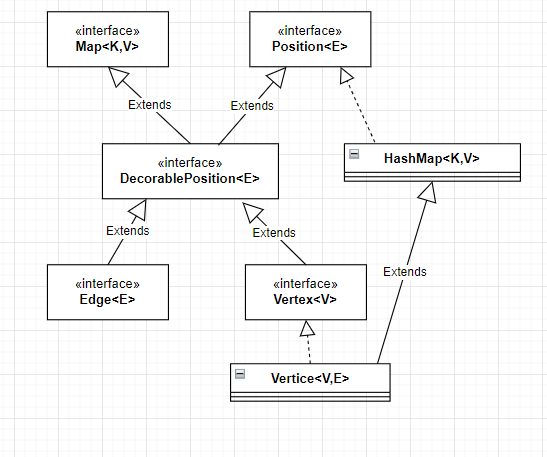
* VENTAJA: En forma abstracta mantengo toda la información del DFS y de futuros algoritmos
* DESVENTAJA: Hay que modificar lo que vayamos programando

Se considera la opción mas viable de las tres

**OPCION N°3: “Decorar vértices del grafo”:**

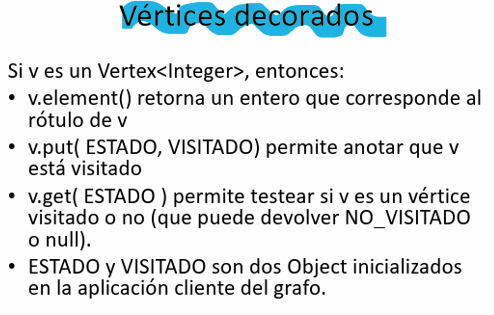
**Diagrama UML de la implementación de vértices decorados de un grafo**

* Una posición decorada es una posición que además es un mapeo

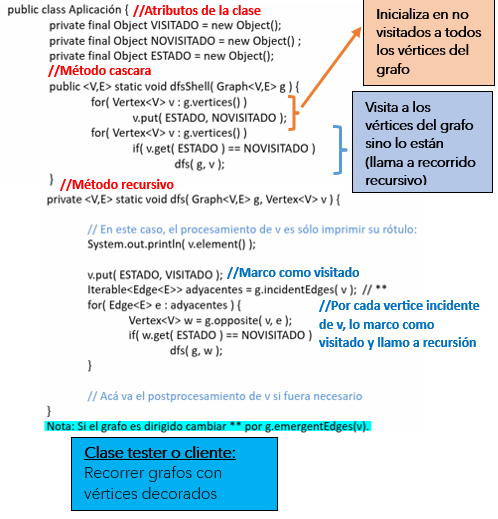


**Creo interfaz DecorablePosition<E>:**

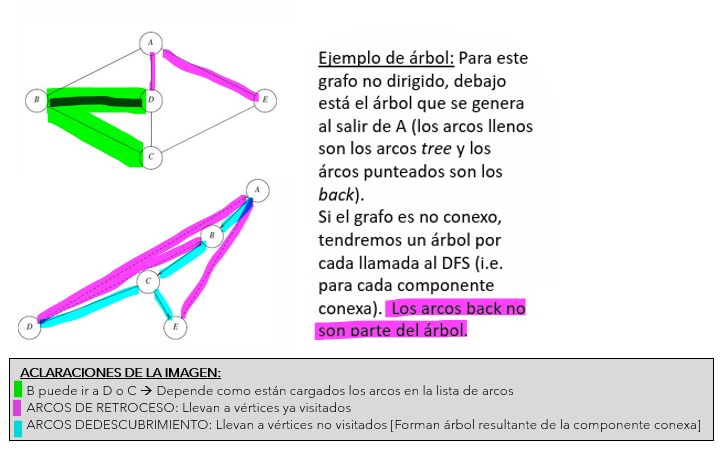
* **Extiende de Position<E>**
* **Extiende de Map<Object,Object>**



**Para implementar los métodos de Map<Object,Object> extiendo a la clase Vertice<V,E> de HashMap<Object,Object>, automáticamente un vertice es un mapeo [hereda métodos]**

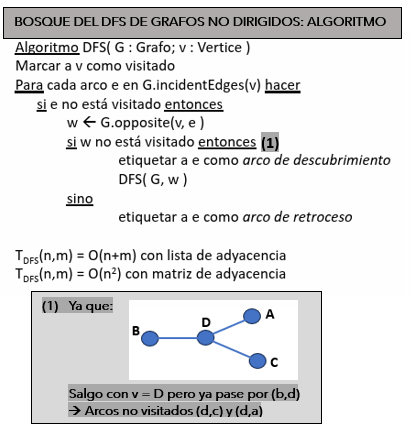


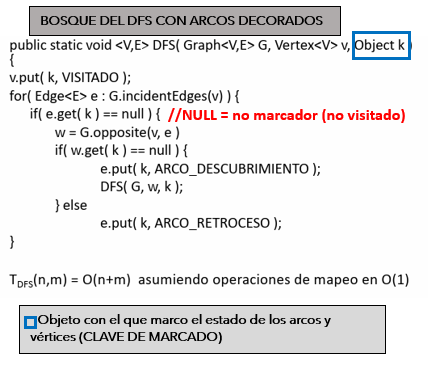
**BOSQUE DEL DFS EN GRAFOS NO DIRIGIDOS (APLICACIÓN DEL RECORRIDO EN PROFUNDIDAD):**

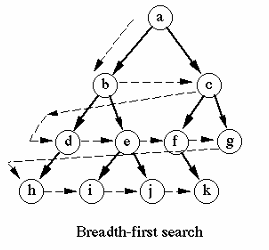
* Las llamadas sucesivas a DFS desde DFSShell van descubriendo vértices que forman árboles
* Estos árboles forman el bosque (o foresta) del DFS
* Al orientar los arcos en la dirección en la que son explorados durante el recorrido, se distinguen:
* **ARCOS DE DESCUBRIMIENTO O ARBOL (Discovery o tree edges):** Arcos que llevan vértices no visitados
* **ARCOS DE RETROCESO (back edges):** Arcos que llevan a vértices que ya fueron visitados (Se descartan)
* Cada vez que llamo desde el método cascara al recursivo, formo un **ARBOL DE RECORRIDO** (con arcos que contienen nodos visitados y no visitados)
* Estos arboles producen mas arboles de recorrido 🡪 Cada árbol formado desde el método cascara se llama **FORESTA**
* **Los arboles forman el GRAFO 🡪 Para un mismo Grafo, voy a tener varios arboles de recorrido**

**APLICACIONES DEL DFS PARA GRAFOS NO DIRIGIDOS EN O(n+m):**

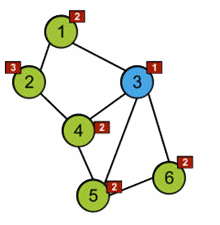
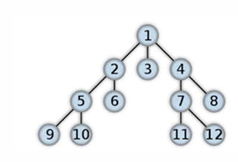
* Testear si G es conexo (todos los vértices quedan visitados si el grafo es conexo)
* Calcular un árbol abarcador si G es conexo (formado por los vértices de G y por sus arcos tree)
* Calcular las componentes conexas (por cada iteración del DFShell incremento un contador indicando el numero de componente conexa y con ese contador etiqueto los vértices de cada componente
* Encontrar un camino entre dos nodos
* Encontrar un ciclo





1. **RECORRIDO EN ANCHURA (BREADTH-FIRST SEARCH o BFS):**

* **Equivale al recorrido por niveles en árboles + un testeo** para no volver a recorrer un subgrafo ya explorado: a, b, c, d, e, f, g, h, i, j, k **(ver imagen)**
* La búsqueda en anchura **(BFS o Breadth First Search)** es un algoritmo para recorrer o buscar elementos en un grafo
* Se comienza eligiendo algún nodo como elemento raíz y se exploran todos los vecinos de este nodo
* A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo
* **APLICACIONES DEL BFS: Todas las del DFS y además: Hallar el camino mas corto (en cantidad de arcos) entre los vértices (en O(n+m))**



**ACLARACIONES DEL RECORRIDO:**

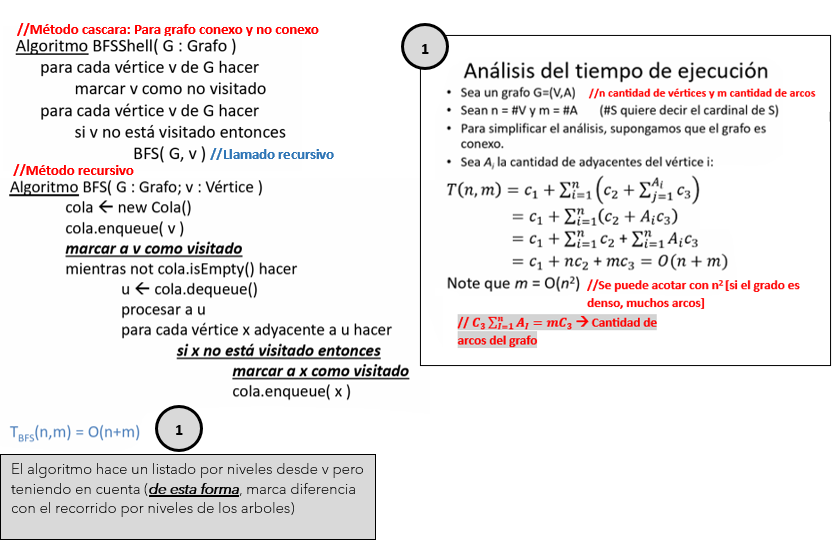
* En el recorrido por nivele, encolo al hijo, si es que no lo vi, a la estructura de la cola auxiliar (BFS)
* El problema central de estos recorridos es cuando el grafo, con el que trabajo, no es un ARBOL (POSEE CICLOS) 🡪 Tengo más de una forma para llegar a un vértice (o alcanzar un vértice)

El recorrido arranca desde el vértice de rotulo 3

En rojo, está la distancia al 3 en cantidad de arcos de acuerdo con el recorrido

Primero arrancamos por el vértice de rotulo 3, luego sigue por los vértices de distancia 1 (1,4,5,6). Luego por el de distancia 2 (2)

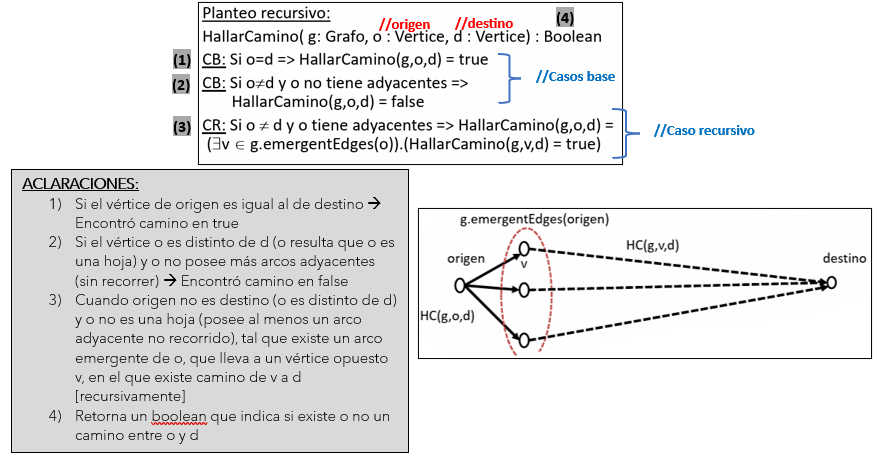
El número del vértice corresponde al orden en el cual es visitado por el algoritmo respectivo

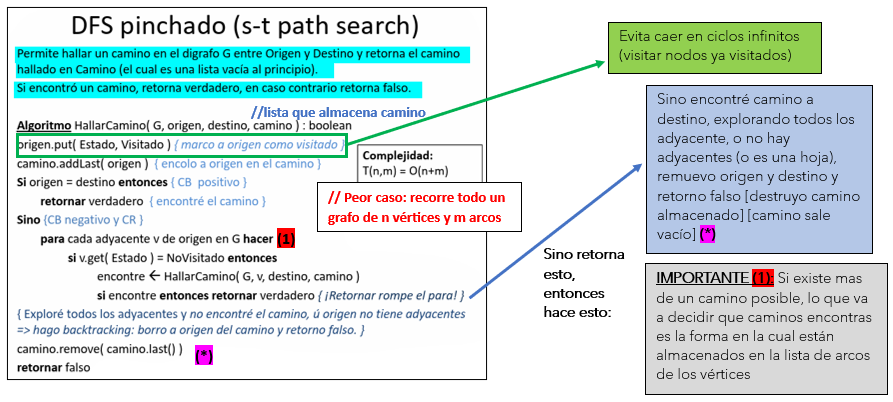


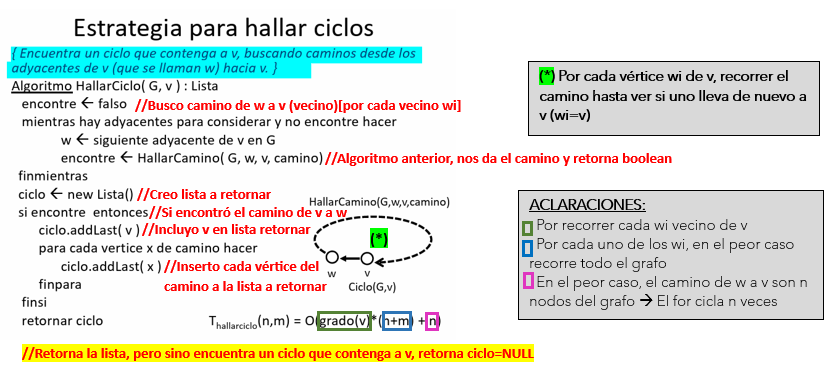
**ALGORITMOS PARA ENCONTRAR CAMINOS EN DIGRAFOS PESADOS CON NUMEROS REALES (CONTINUACION DE RECORRIDO DE GRAFOS):**

1. **DFS pinchado (st-path search):** Permite encontrar un camino entre dos vértices s y t
2. **BFS:** Para hallar camino con cantidad mínima de arcos
3. **DFS con backtracking , marca y desmarca:** Permite encontrar un camino de costo mínimo entre dos vértices s y t
4. **Dijkstra:** Permite hallar todos caminos de costo mínimo entre un vértice a y todos los otros vértices
5. **Floyd:** Permite hallar el camino de costo mínimo entre cada par de vértices s y t

**ST-PATH SEARCH:**

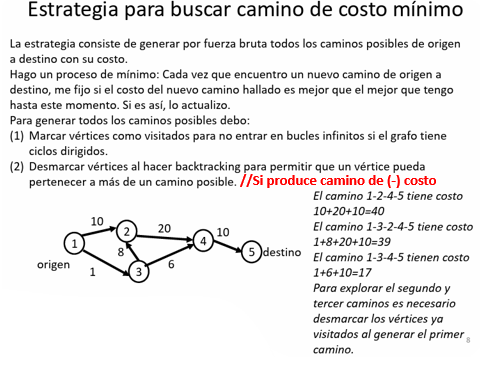
* **PROBLEMA:** Buscar un camino en un grafo G de un vértice s a un vértice t
* **S:** Origen
* **T:** Destino
* **ESTRATEGIA:** Explorar el grafo en DFS manteniendo el camino explorado y marcando los vértices visitados para no volver a explorar porciones del grafo ya explorados



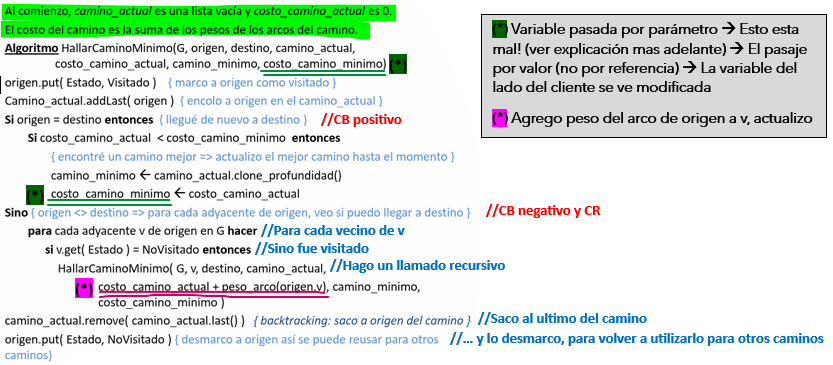


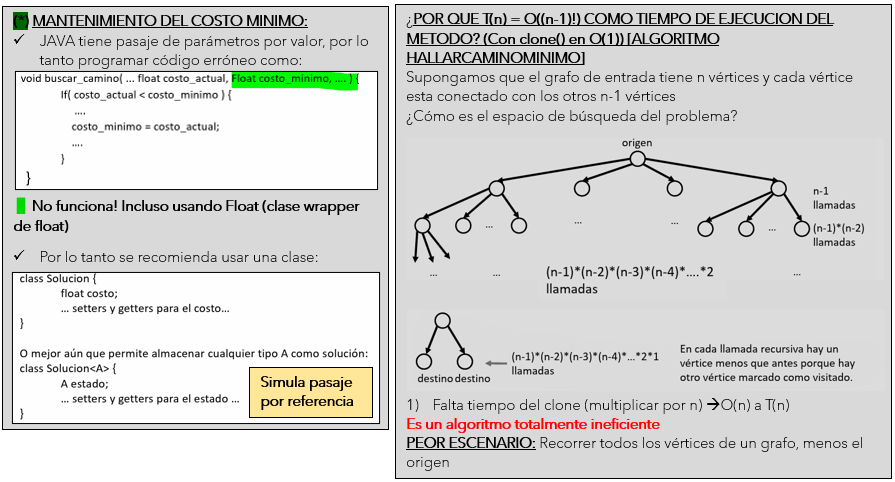
**DFS CON MARCA Y DESMARCA:**

* Dado un dígrafo pesado con números reales, permite hallar un CAMINO DE COSTO MINIMO entre dos vértices origen y destino computando el camino y su costo (entendido como la suma de los pesos de los arcos)
* El tiempo de ejecución para un grafo que tiene todos los arcos entre cada par de nodos es O(n!) [ n! se da en el peor caso que es cuando todos los vértices están conectados con todos los otros vértices) [TOTALMENTE INEFICIENTE]



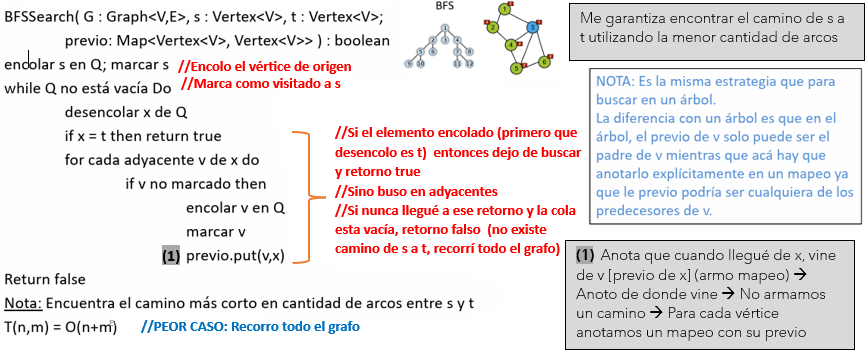


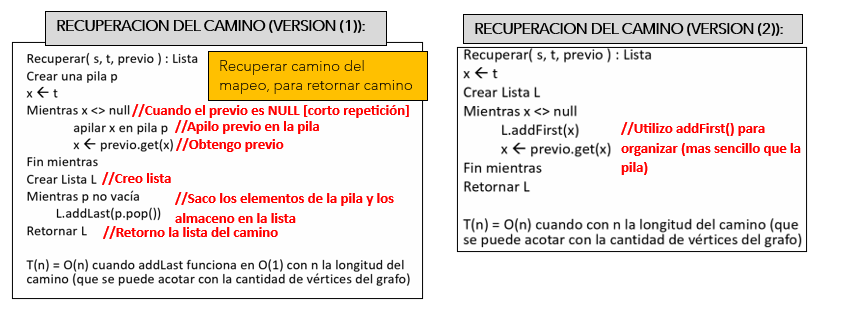




**BFS PARA HALLAR CAMINOS:**

* Retorna true si hay camino de s a t en un grafo G
* Previo es un mapeo tal que previo.get(v) retorna el nodo previo de v en el camino de s a v

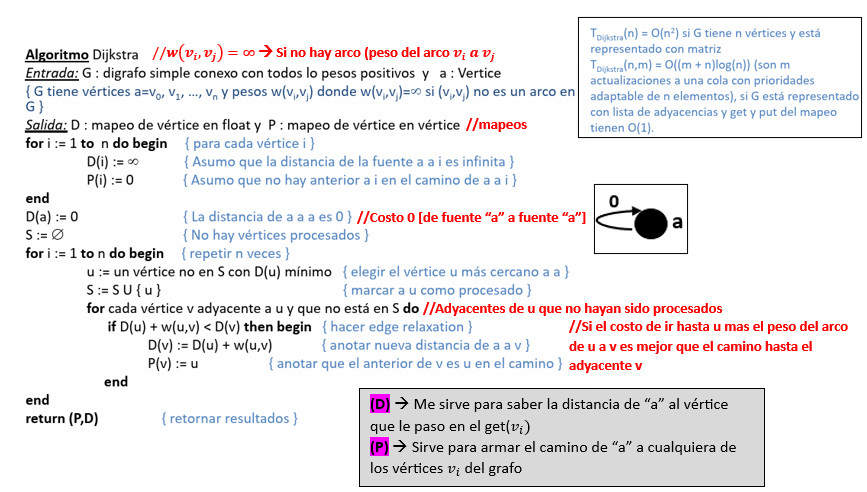


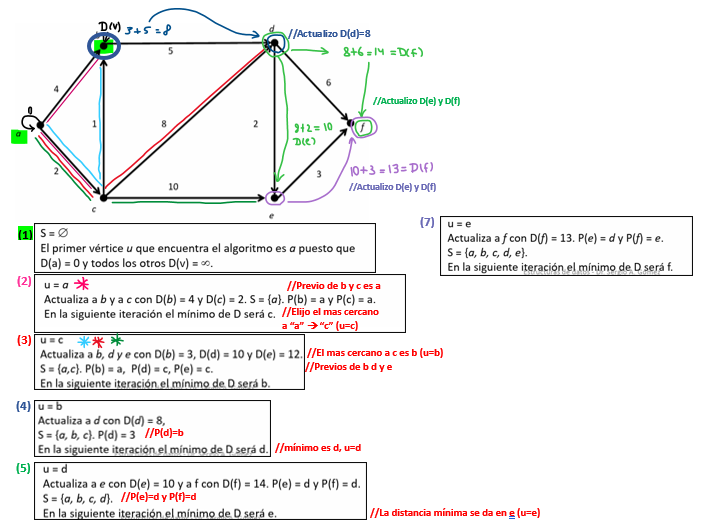


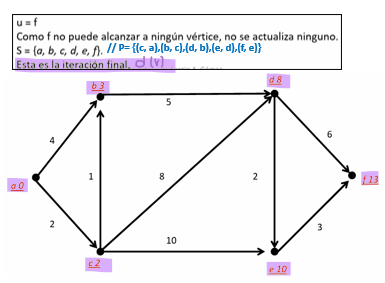
**//Agregar papel con explicación del recorrido**

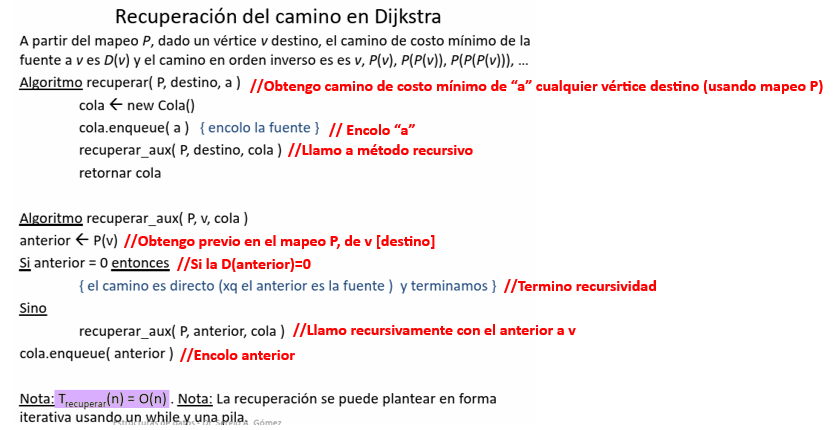
**ALGORITMO DE DIJKSTRA:**

* Suponga un dígrafo G tal que cada arco tiene costo no negativo, Dijkstra computa los caminos de costo mínimo desde un vértice a “a” todos los otros vértices de G
* El vértice “a” se conoce como la fuente
* El COSTO DEL CAMINO es la suma de los pesos de los arcos del camino
* El algoritmo mantiene un conjunto S con los vértices cuyo camino con la distancia más corta es conocida [mínima] (S inicialmente está vacía)
* En cada paso se agrega a S el vértice u cuya distancia a la fuente es tan cercana cómo es posible
* El algoritmo termina cuando S contiene todos los vértices
* La salida del algoritmo son dos mapeos D : V 🡪 Float y P: V 🡪 V tal que:
* D(v) es la distancia a v desde la fuente
* P(v) es el vértice anterior a v en el camino desde la fuente a v





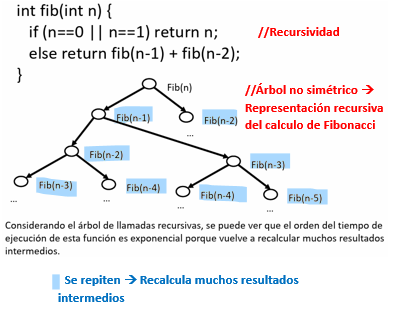


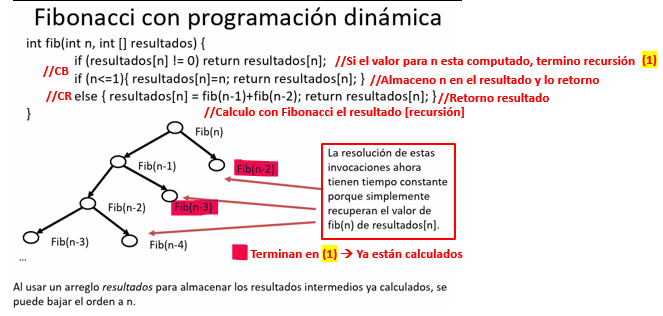


**PROGRAMACION DINAMICA:**

**La programación dinámica es un método para resolver un problema dividiéndolo en una colección de subproblemas menores, resolviendo esos subproblemas sólo una vez, y almacenando sus soluciones usando una estructura de datos en memoria principal (e.g. arreglo, mapeo, etc.)**

**//PROGRAMACION DINAMICA**

**ALGORITMO FIBONACCI:**



**ALGORITMO WARSHALL:**

1. **COMPUTO DE LA CLAUSURA TRANSITIVA R\* DE UNA RELACION BINARIA R:**

**//Ejemplo: Grafo de la relación divide**

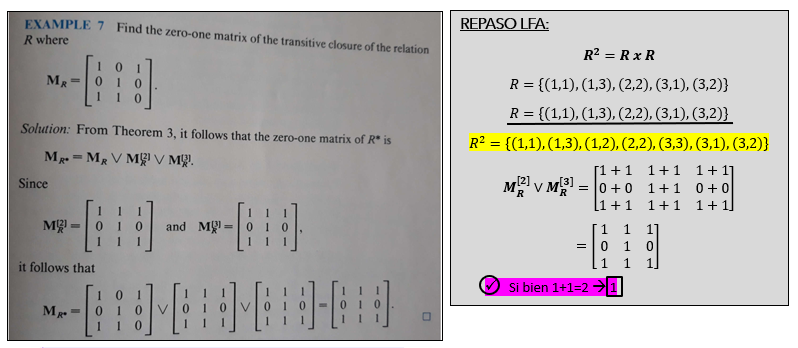
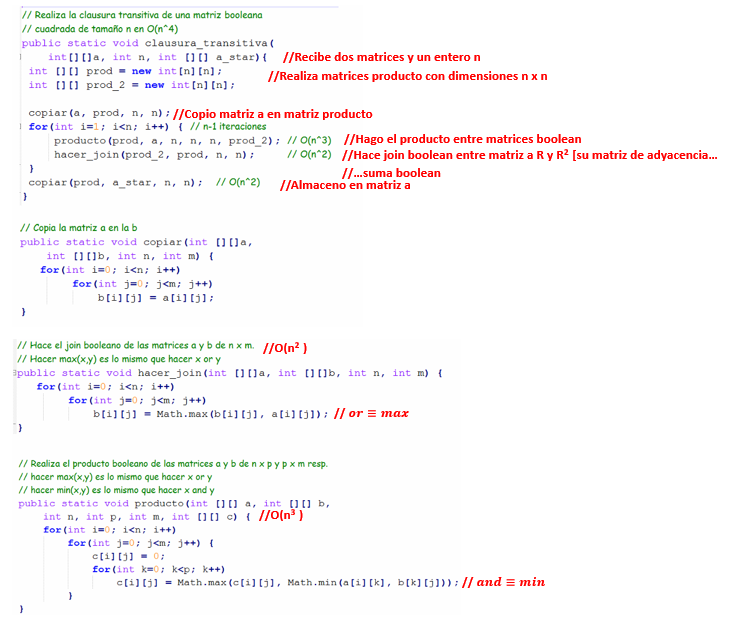
* Una relación binaria R es transitiva cuando aRb y bRc implica aRc
* Dado un grafo que representa una relación R, el algoritmo de Warshall permite computar la clausura transitiva de R, notada como R\*
* La clausura transitiva de R es la relación transitiva más pequeña que contiene a R
* Cuando R es representada con un grafo G dirigido, si R no es transitiva, entonces G no contiene todos los arcos para los vértices que pueden ser unidos mediante caminos

**EJEMPLO:** R = { (1,2), (2,3) } entonces R\* = R U {(1,3)} pues es posible ir de 1 a 3 (pasando por 2

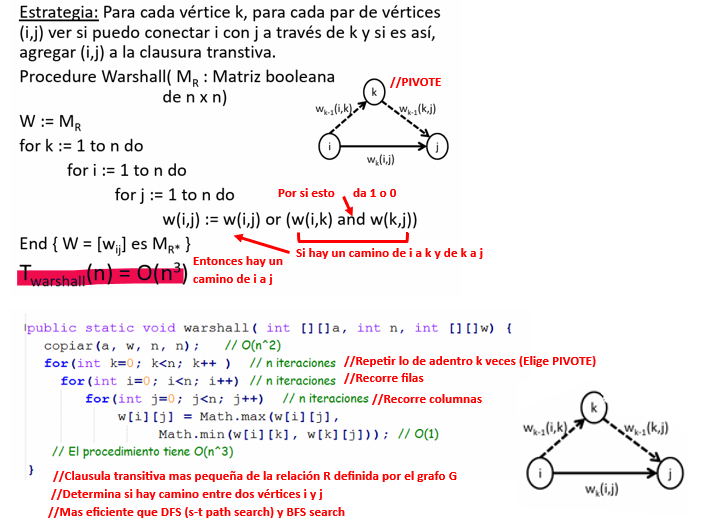
1. **COMO CALCULAR R\*:**

* Si la relación R entre n elementos se representa con una matriz booleana de n x n (booleana quiere decir formada por 1s y 0s), entonces (es decir, R x R x R x…xR realizado n veces, con “x” representando el producto booleano de matrices)
* La clausura R\* se calcula como R U R2 U R3 U …. U Rn, donde U representa el join-booleano (or-booleano componente a componente) entre matrices

**//Unión de matrices de adyacencia**

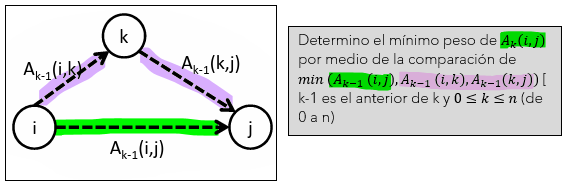


**Algoritmo de warshall en lenguaje C**



**ALGORITMO DE WARSHALL (PROGRAMA)**

**ALGORITMO DE FLOYD: “CAMINOS MINIMOS”**

* Dado un digrafo pesado G=(V,A) donde cada arco tiene un peso numérico no negative
* Queremos calcular los caminos de costo mínimo de todos los vértices a todos los vertices
* Supongamos que C(i,j) es peso del arco (i,j) de A
* Usaremos una matriz A de n x n tal que inicialmente A(i,j)=C(i,j), o si no hay arco entre i y j
* En la iteración k-ésima veremos si actualizamos a A de acuerdo a
* P(i,j) almacenerá a k (i.e., uno de los vértices intermedios en el camino de i a j)

La matriz restante muestra costo mínimo de los caminos de todos los vértices a todos los vértices

