

GRAFOS

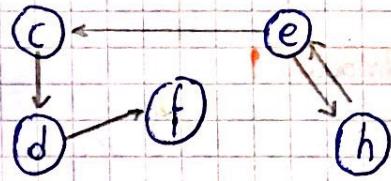
Un grafo es un conjunto de objetos llamados vértices/nodos unidos por enlaces llamados aristas/arcos que permiten relaciones binarias entre los elementos del conjunto. Gráficamente se representa como un conjunto de nodos unidos por líneas.

$G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de pares donde (u, v) es un par para el cual $v \in V$ y $u \in V$.

GRAFO DIRIGIDO

Los arcos tienen direcciones definidas y por lo tanto $(u, v) \neq (v, u)$. La relación sobre V no es simétrica.

Un grafo dirigido puede ser fuertemente conexo si existe un camino desde cualquier vértice hacia cualquier otro vértice.



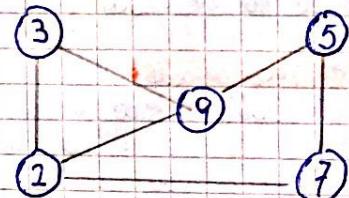
$$G(V, E) \quad V = \{c, d, f, e, h\}$$

$$E = \{(c, d), (d, f), (e, c), (e, h), (h, e)\}$$

GRAFO NO DIRIGIDO

Los arcos no tienen una dirección definida x lo tanto $(u, v) = (v, u)$.

En un grafo no dirigido dos vértices se dicen adyacentes si existe un arco que une esos dos vértices.



$$G(V, E) \quad V = \{2, 3, 9, 7, 5\}$$

adyacencia que algo sea adyacente significa que está muy próximo/unido a otra cosa $E = \{(2, 3), (2, 7), (2, 9), (3, 9), (5, 7), (5, 9)\}$

- v es adyacente a u si hay un arco que los une

- en un grafo no dirigido $(u, v) \in E$ incide en los arcos c, v

- en un grafo dirigido $(u, v) \in E$ incide parcialmente en u, v

grados

· grafo no dirigido: nro de aristas que inciden en él

· grafo dirigido: hoy en grafo de sólido (grado-out) y uno de entrada (grado-in)

· grado-out: nro de aristas que parten de él

· grado-in: nro de aristas que inciden en él

· el grado del vértice es la suma de grado-out + grado-in

· grado de grafo: máximo grado de sus vértices

caminos

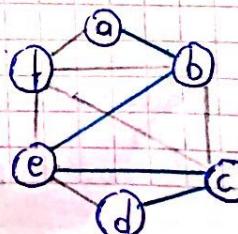
el camino desde v a u es una secuencia $v_1, v_2, v_3, \dots, v_k$ tal que $v_1 = v$ y $v_k = u$

el camino de a a d es $\langle a, b, e, c, d \rangle$

longitud del camino: nro de aristas del camino.

la longitud del camino de a a d siendo

$\langle a, b, e, c, d \rangle$ es 4

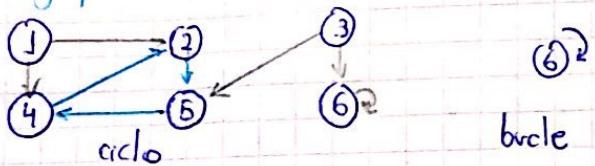


-camino simple: camino en el que todos sus vértices excepto el 1º y el último son distintos

-ciclo: caminando desde v_1, v_2, \dots, v_k tal que $v_1 = v_k$

-bucle: ciclo de longitud 1

-grafo acíclico: grafo sin ciclos

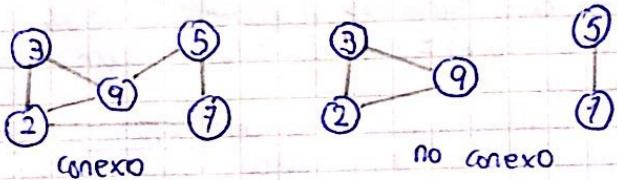


grafo ponderado: el arco tiene asociado un valor / etiqueta

conectividad

grafo dirigido conexo

tiene un camino p/c por de vértices



bosque: grafo sin ciclos

árbol libre: bosque conexo

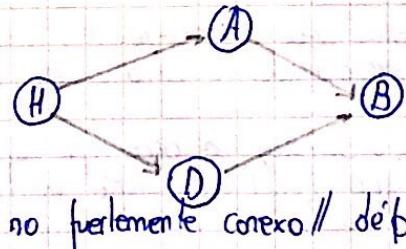
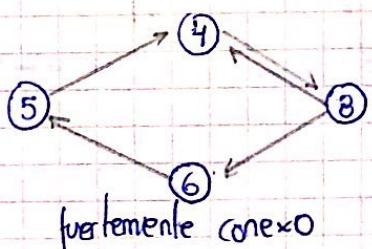
árbol: árbol libre en el que un nodo se ha designado como raíz

conectividad en grafos dirigidos

v es alcanzable desde u si hay un camino de u a v

es fuertemente conexo si hay un camino desde cualquier vértice a cualquier otro vértice.

si un grafo dirigido no es fuertemente conexo pero el grafo subyacente (sin sentido de aristas) es conexo, el grafo es débilmente conexo.



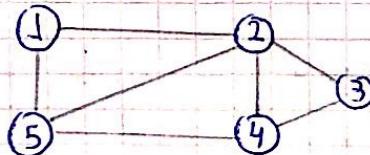
representaciones

los grafos pueden representarse mediante matriz de adyacencia o lista de adyacencia. debo tener en cuenta que los arcos pueden o no tener peso, los arcos pueden ser o no dirigidos, se pueden permitir o no varios arcos entre los mismos nodos y se pueden permitir o no arcos hacia el mismo nodo

MATRIZ DE ADYACENCIA

dado un grafo $G = (V, E)$, podemos representar un grafo o través de una matriz de $V \times V$ donde el contenido de cada casilla será un valor binario (0,1) el cual indicará si (u,v) son adyacentes. Entonces

$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{si no} \end{cases}$$



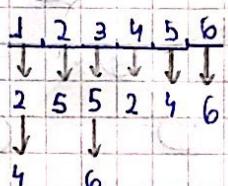
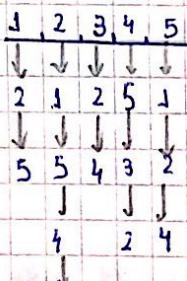
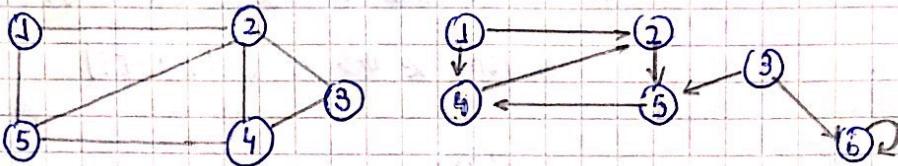
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

* p/ grafos dirigidos solo marcaré los adyacentes de aristas-in

* p/ grafos pesados marco el peso p/c arista

LISTA DE ADYACENCIAS

se realiza usando un vector de listas, formando un índice de vértices. De cada elemento de índice parte una lista encadenada describiendo los vértices adyacentes.



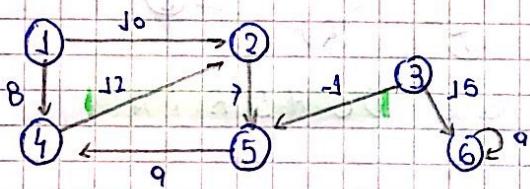
} lista solo' sus vértices de salida

grafo dirigido.

* grafo pesado: c/ marca tanto' el valor del vértice y el peso de
el costo de conectarlo con lo que lo conecta.

propiedades: si G es dirigido, la suma de las longitudes de la lista de adyacencia será $|E|$

si G es no dirigido, la suma de la lista de adyacencia será $2|E|$
el costo de comprobar si uno existe (u,v) es adyacente sera de $O(|grob G|)$



grafo pesado.

recorridos

en muchas aplicaciones es necesario visitar todos los vértices del grafo o partir de un nodo dado.
hay 2 enfoques básicos

DFS (depth-first search / recorrido en profundidad): la idea es olfatearse lo más posible del nodo inicial (sin repetir nodos), luego devolverse un paso e intentar lo mismo x otro camino.

es una generalización del recorrido preorden de un árbol. Funciona de lo siguiente:

- parte del vértice v
- cuando se visita un nuevo vértice, exploro el camino q' salgo de él
- hasta que no haya finalizado de explorar uno de los caminos no se comienza c/ el sig
- un camino dejó de explorarse cuando llegó a un vértice visitado.

!! si existen vértices no alcanzables desde v , el recorrido quedará incompleto entonces debo seleccionar algún vértice como nuevo punto de partida y repetir el proceso.

PSEUDOCÓDIGO

```
dfs (V: vértice)
    marco [V] := visitado;
    para todo nodo w adyacente a V
        si w no está visitado
            dfs(w);
```

main: dfs (grafo)

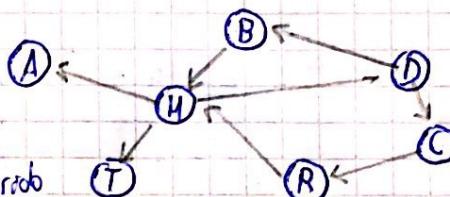
inicializa marca en falso
para cada vértice del grafo
si v no está visitado
dfs(v);

tomando D como pto de partida, el recorrido
será **D C R H T A B**

TIEMPO DE EJECUCIÓN

$G(E, V)$ se representa c/ una lista de adyacencia.
el método $dfs(v)$ es aplicado en nodos no visitados y solo una vez x vértice, d/ tiempo de todos los llamados a $dfs(v)$: $O(|E|)$
o sea debes sumar el tiempo de main: $O(V)$

∴ el $T(n)$ de DFS es $O(|V| + |E|)$



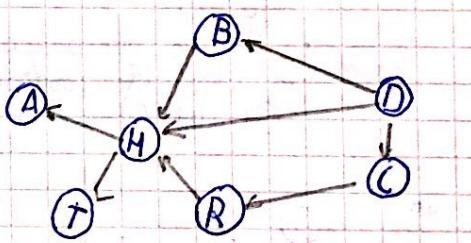
BFS (breadth-first search / recorrido en amplitud): visita a todos los vecinos del nodo inicial, luego a los vecinos de los vecinos y así sucesivamente. Resulta ser una generalización del recorrido x niveles de un árbol. Funcionando así:

- parte de un vértice u, visita u y después visita los nodos adyacentes a él
- repite el proceso p/ c/u de los adyacentes a u, siguiendo el orden en el que se visitaron

ESQUEMA ITERATIVO

dado $G(V, E)$

- encolo el vértice origen
- marco u como visitado
- proceso la cola
- desencola u y p/ v adyacente a u (u, v), si v no ha sido visitado, encolo y visita v



sólida **D C H B R T A**

DFS / BFS $O(n)$

ÁRBOLES DE EXPANSIÓN DE DFS

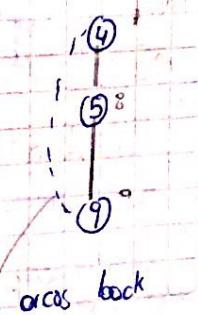
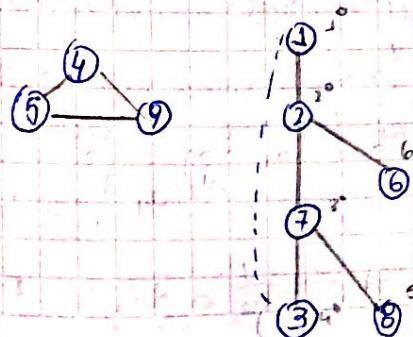
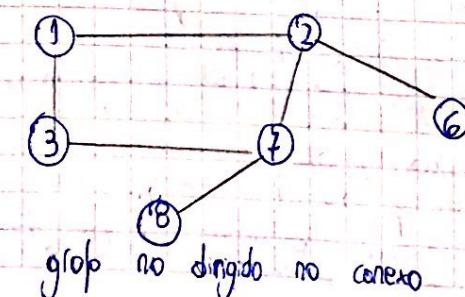
el recorrido DFS representa un árbol c/ aristas y vértices que recorrió. El recorrido no es único pq depende del vértice inicial y del orden de visita de sus adyacentes. Puedo obtener

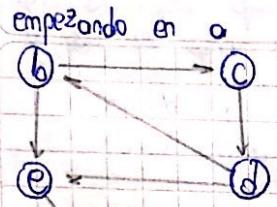
- un único árbol (árbol de expansión)

- más de uno (bosque de expansión). Esto ocurre cuando

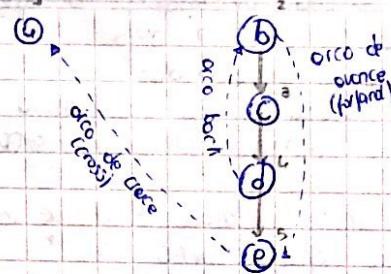
1) el grupo es no dirigido y no conexo

2) el grupo es dirigido y no fuerte conexo (depende del vértice que tome como pto. de partida)





grado dirigido y no fuertemente conexo



CLASIFICACIÓN DE LOS ARCOS EN UN GRAFO DIRIGIDO EN BOSQUE DE EXPANSIÓN DFS

tree = arcos del bosque DFS, arcos que conducen a vértices no visitados en la búsqueda forward = son arcos $v \rightarrow v$ que no están en el bosque, donde v es descendiente pero no es hijo en el órbol

backward = son los arcos $v \rightarrow v$, donde v es antecesor en el órbol. Un arco de un vértice o si mismo es considerado como arco back. Son los arcos que retornan al vértice inicial (crea un círculo)

cross = son todos los arcos $v \rightarrow v$, donde v no es ni antecesor ni descendiente de u . Son arcos que parten de v a vértices del mismo órbol o a vértices de otros órbols en el bosque DFS

ALGORITMO DE KOSARAJU

el algoritmo de Kosaraju es un algoritmo lineal usado para encontrar componentes fuerte conexos en un grafo dirigido

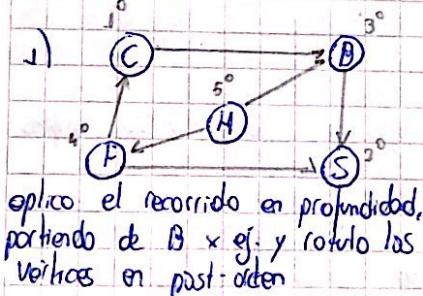
Algoritmo aplicar DFS(G) rotulando los vértices de G en post-orden (apilar en la pila) construir el grafo reverso, o sea G^R (inverter los aristas)

aplicar DFS(G^R) comenzando x los vértices de G rotulado (tope de pila) cada órbol de expansión resultante del 3º paso es una componente fuerte conexa

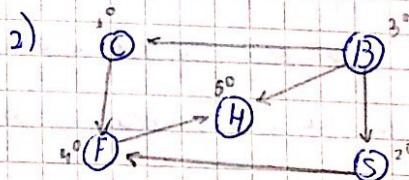
!! apilaré cuando no pueda avanzar o sus adyacentes ya están visitados

EJEMPLO I

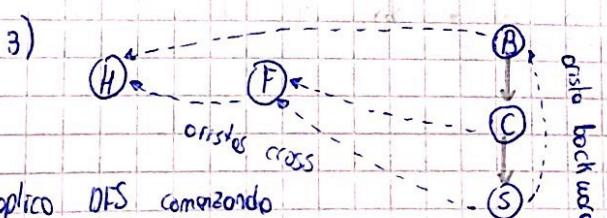
H	5°
F	4°
B	2°
S	2°
C	1°



aplico el recorrido en profundidad, partiendo de B x ej. y rotuló los vértices en post-orden



construyo el grafo reverso G^R



obtuve 3 componentes fuerte conexas.

aplico DFS comenzando de los vértices de mayor rotulado en G^R

en Kosaraju se realizan 2 DFS y recorro todos los aristas una vez $O(|V| + |E|)$

resulta ser un listado de los vértices

ordenación topológica

la ordenación topológica es lo mismo en lo que ordenamos un digraf. Completa los sig. prop
- si G contiene un ciclo dirigido entonces el nodo v aparece antes del nodo u
- el G NO DEBE ser cíclico

- la ordenación topológica no es única

la ordenación topológica puede ser usada al la precedencia de eventos/planeaciones
y/o ordenación hoy <→ técnicas que varían en su eficiencia

el gráfico debe tener al menos un vértice cuyo grado-in = 0 de lo contrario significa q' hoy en círculo y al menos habrá un vértice cuyo grado-out = 0

VERSIÓN I

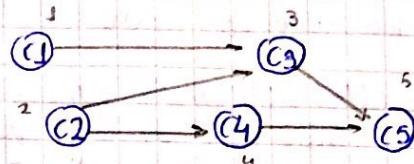
el algoritmo usa un arreglo-in en el qie se almacenan los grados de entrada de los vértices y en cada paso se toma de allí un vértice de grado-in = 0

pasos globales:

- 1- seleccionar un vértice v con grado-in = 0
- 2- visitar v
- 3- dejar de tener en cuenta el vértice v y disminuir en 1 el grado-in de los vértices adyacentes al vértice v
- 4- repetir el paso 1 hasta seleccionar todos los vértices

$$O(NV^2 + IEI)$$

si lo piso me queda vacío :: hoy círculo



$$c_1, 0, c_3, c_4, c_5 \Leftarrow \text{grado-in}$$

$$0 \ 0 \ 2 \ 1 \ 2$$

$$0 \ 0 \ 1 \ 1 \ 2$$

$$0 \ 0 \ 0 \ 0 \ 2$$

$$0 \ 0 \ 0 \ 0 \ 1$$

$$0 \ 0 \ 0 \ 0 \ 0$$

$$\text{salida } c_3 \ c_2 \ c_3 \ c_4 \ c_5$$

VERSIÓN II

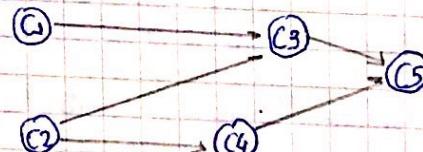
esta versión el algoritmo usa un arreglo grado-in en el qie se almacenan los grados de entrada de los vértices y una pila (Q o cola Q) en donde se almacenan los vértices con grados de entrada igual a 0

pasos globales:

- 1- almacenar en el arreglo grado-in los grados-in de c/ vértice
- 2- aquellos vértices cuyo grado-in = 0, los apilo
- 3- medida qie desapila, decremento grado-in de c/ adyacente
- 4- repite paso 2 hasta qie la pila queda vacía.

si en algún momento la pila queda vacía y todavía no todo el arreglo no está en 0 ∴ significa q' hoy en círculo.

$$O(NV + IEI)$$



$$c_1, c_2, c_3, c_4, c_5$$

$$0 \ 0 \ 2 \ 1 \ 2$$

$$0 \ 0 \ 1 \ 0 \ 2$$

$$0 \ 0 \ 0 \ 0 \ 1$$

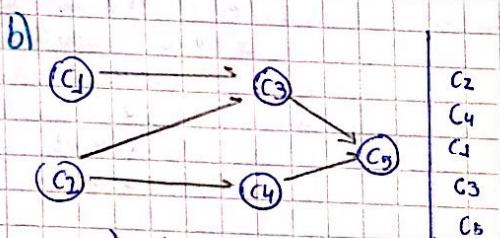
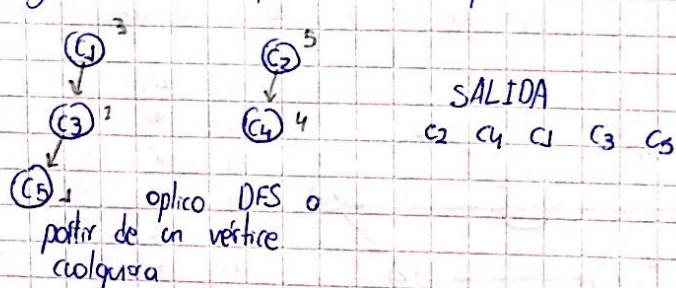
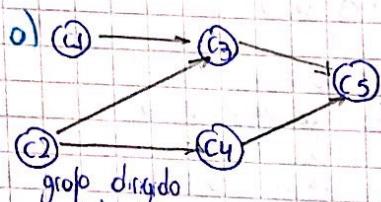
$$0 \ 0 \ 0 \ 0 \ 0$$

$$\text{salida } c_2 \ c_4 \ c_3 \ c_5 \ c_5$$

VERSIÓN III

en esta versión se aplica el recorrido en profundidad.
se realiza un recorrido DFS, marcando el vértice en post-orden, es decir, una vez visitados todos los vértices a partir de uno dado, el marcado de los vértices en post-orden puede implementarse si las sig. opciones:

- numerándolos antes de retocarlos en el recorrido; luego se listan los vértices si sus números de post-orden de mayor a menor
- colocándolos en una pila P , luego se listan empezando x el tope.



- aplico DFS o partir de un vértice cualquiera x ej. C1 y opilo los vértices en post-orden
- listo los vértices o medida q' los desapilo

$O(n)$

!! en DFS cuando llego a un vértice que no me deja continuar y aún hay vértices x recorrer entonces dobo elegir uno no visitado y continuar x ahí

camino de costos mínimos

el camino de costo mínimo desde un vértice v_i a otro vértice v_j es aquel en que la suma de los costos de los aristas sea mínima
sea $G = (V, A)$ un grafo dirigido y pesado, el costo de $c(i, j)$ está asociado a la arista $v(i, j)$. Dado un camino: $v_1, v_2, v_3, \dots, v_n$, el costo del camino es

$$C = \sum_{i=1}^{N-1} c(i, i+1)$$

C también se conoce como longitud del camino pesado
la longitud del camino no pesado es la cantidad de aristas

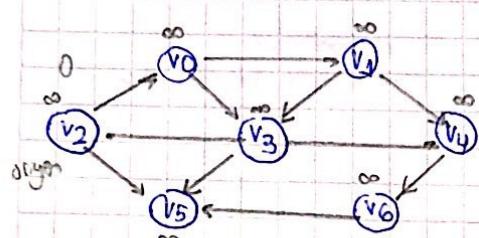
ALGORITMOS calculan los caminos mínimos desde un vértice origen s a todos los restante vértices del grafo

GRAFOS SIN PESOS

- Pl c/ vértice v se mantiene lo sig. información
- D_v : es la distancia mínima desde el origen s (inicial) o p/ todos los vértices excepto el origen (v / valor 0)
 - P_v : vértice x donde pasó p/ llegar
 - conocido: dato booleano q' me indica si está procesado

ESTRATEGIA → recorrido en amplitud (BFS)

- 1- avanzar x niveles a partir del origen, asignando distancias s/ se alcanza (se usa uno colo)
- 2- inicial, s $D_s = \infty$. Al inspeccionar v se reduce al valor correcto. $D_v = D_s + 1$
- 3- desde c/ v , visitamos a todos los nodos adyacentes a v



V_i	D_s	D_v	conoc
v_0	∞	0	0
v_1	∞	0	0
v_2	0	0	0
v_3	∞	0	0
v_4	∞	0	0
v_5	∞	0	0
v_6	∞	0	0

algoritmo

poro cada vértice $v \in V$

$D_v = \infty; P_v = 0; Conoc_v = 0;$

$D_s = 0; encola(Q, s); Conoc_s = 1$

mientras (not encola(Q)) {

desencolar(Q, u);

pl c/ vértice $w \in V$ adyacente a u y

si (w no es conocible) {

$D_w = D_u + 1;$

$P_w = u;$

encola(Q, w);

Conoc_w = 1;

GRAFOS CON PESOS POSITIVOS

ESTRATEGIA → algoritmo de Dijkstra

- 1) dado un vértice origen si elegir el vértice v que esté a menor distancia de s , dentro de los vértices no procesados
 - 2) marcar v como procesado
 - 3) actualizar la distancia de w adyacente a v
- Pl c/ vértice v mantiene lo siguiente información
- D_v : distancia mínima desde el origen/inicial o p/ \neq vértices
 - P_v : vértice p/ diente paso p/ llegar
 - conocido: dato booleano q' me indica si está procesado (inicial todos en 0)

lo actualizan de la distancia de los adyacentes w se realizó c/ el sy. criterio:
se compara D_w con $D_v + c(v, w)$ y actualiza si $D_w > D_v + c(v, w)$

distancia de s a w
(sin paso p/ v)

distancia de s a
 w , pasado x v

funciona cuando marca vértice como conocido pq' digo q' no hay camino mejor

GRAFOS CON PESOS POSITIVOS Y NEGATIVOS

ESTRATEGIA → encolar los vértices

Si el grafo tiene aristas negativas, el algoritmo de Dijkstra puede dar en rtos errores

- 1) encolar el vértice origen s
- 2) procesar lo colo:

- desencolar el vértice
- actualizar la distancia de los adyacentes w siguiendo el mismo criterio Dijkstra
- si w no está en lo colo, encolalo

O(|V| |E|)

GRAFOS ACÍCLICOS

ESTRATEGIA → orden topológico

resulta ser uno optimizox de Dijkstra.

la selección de vértice se realiza siguiendo el orden topológico.

esta estrategia funciona correcto, dado que al seleccionar un vértice v_i , no se va a encontrar una distancia de menor, pq ya se procesaron todos los caminos q'co llegan a él

$$O(|V| + |E|)$$

CAMINOS E/ TODOS LOS PARES DE VÉRTICES

ESTRATEGIA → algoritmo de Floyd

Habrá 2 matrices D y P , ambas de $|V| \times |V|$

matriz de costos mínimos

matriz de vértices intermedios

$$(|V|^3)$$

GRAFOS
No pesados
Pesados
Pesados negativos
pesados cíclicos

BFS $O(V+E)$
óptimo
incorrecto
incorrecto
incorrecto

Dijkstra
correcto
óptimo
incorrecto
correcto

Encolo vértices
malo
malo
óptimo
malo

Optimizox Dijkstra
incorrecto (1 círcos
"
óptimo

correcto → adecuado pero no es el mejor
malo → solva muy lento