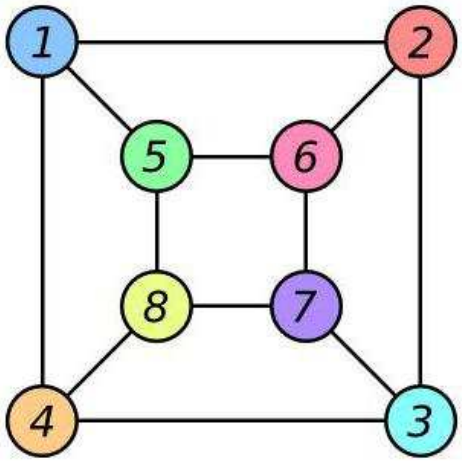
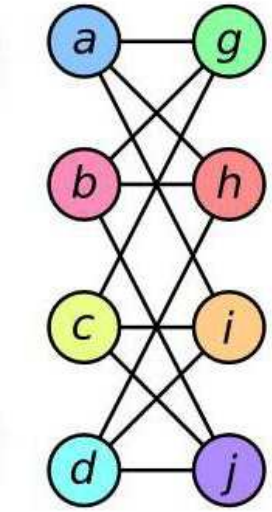
Algoritmos y Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán

2024

Grafo(3)

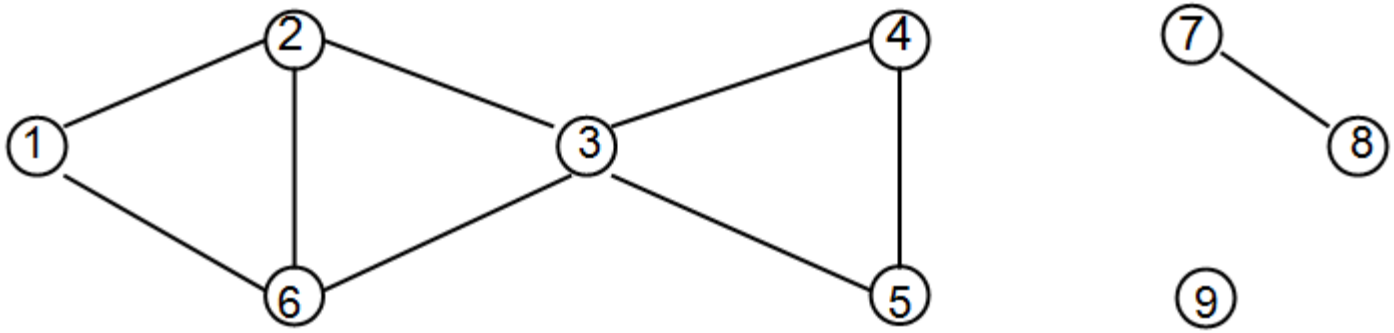




Definiciones

Se van a detallar las siguientes definiciones:

- cadena en un grafo / camino en un digrafo
- longitud de cadena / longitud de camino
- cadena simple / camino simple
- ciclo / circuito
- ciclo simple / circuito simple
- grafo conexo / digrafo fuertemente conexo
- dag : digrafo acíclico
- árbol , bosque



Camino/cadena

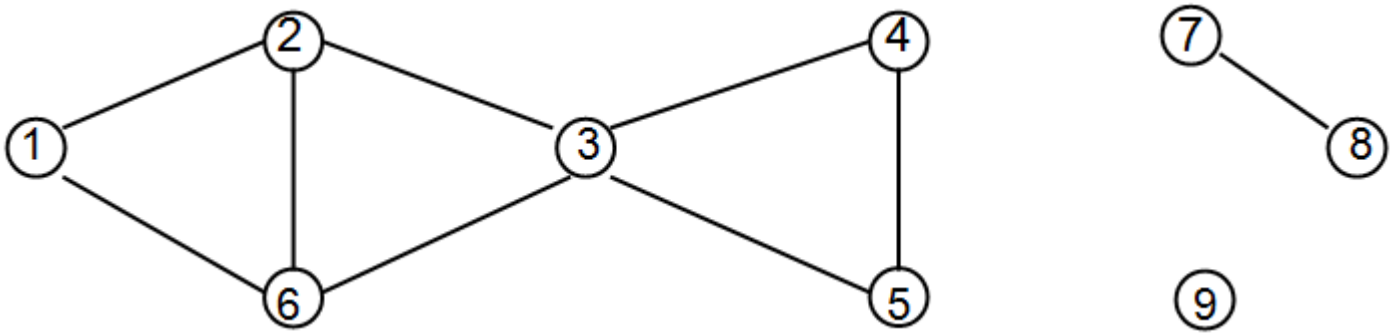
Un ***camino* en un digrafo** o ***cadena* en un grafo** es una secuencia finita de vértices unidos por arcos (aristas) tal que:

- comienza y termina con vértices,
- los arcos no aparecen repetidos,
- cada arco parte del vértice que lo precede y llega al vértice que lo sucede.

Se denota con: $< v_0 , v_1, v_2, ..., v_k >$

Se dice que el camino parte del primer vértice v_0 y llega al último v_k.

Ejemplos de caminos: <1,2,3,4,5,3,6,1>, <3,2,6,3,4,5>, <7,8>



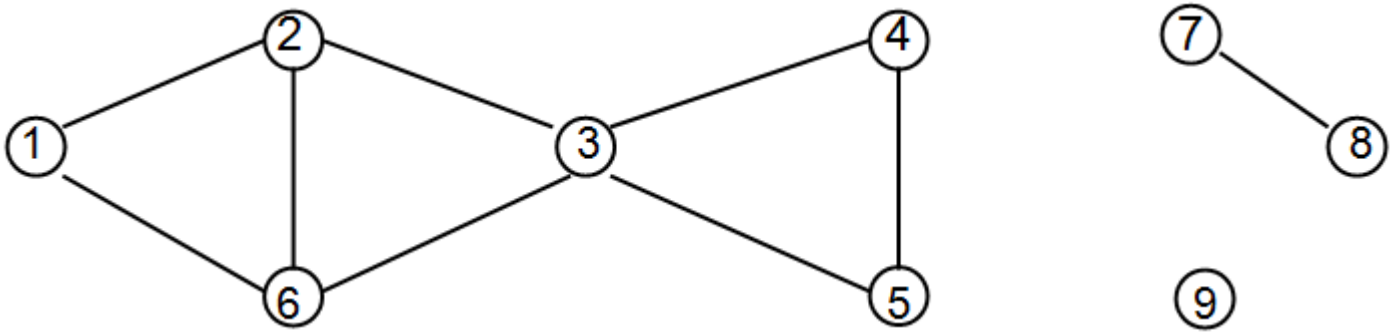
Longitud de Camino

La ***longitud del camino*** es el número de arcos en ese camino.

Longitud del camino < v_0 ,v_1,v_2,...,v_k> = k

Ejemplos de longitud de camino:

- longitud de camino <1,2,3,4,5,3,6,1> = 7
- longitud de camino <3,2,6,3,4,5> = 5
- longitud de camino <7,8> = 1

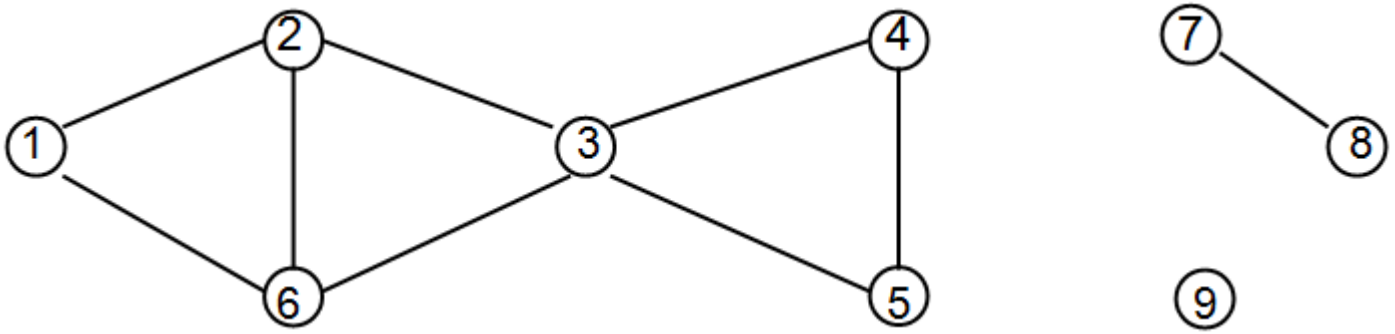


Camino Simple

El ***camino es simple*** si todos sus vértices, excepto el primero y el último son distintos.

Ejemplos:

<1,2,3,4,5>
<4,5,3,6>
<2,3,6,1,2>
<3,4,5,3>



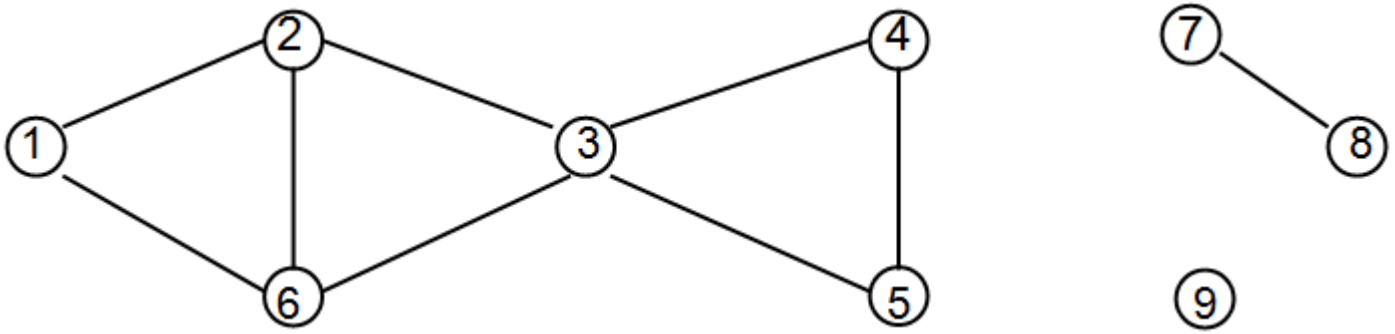
Circuito/ciclo

Un ***circuito o ciclo*** es un camino en el que coinciden el primero y el último vértice.

- Ejemplo: <1,2,3,4,5,3,6,1>

Un ***circuito simple o ciclo simple*** es un camino simple en el que coinciden el primero y el último vértice.

- Ejemplos: <4,5,3,4> <2,3,6,1,2>



Grafo Conexo

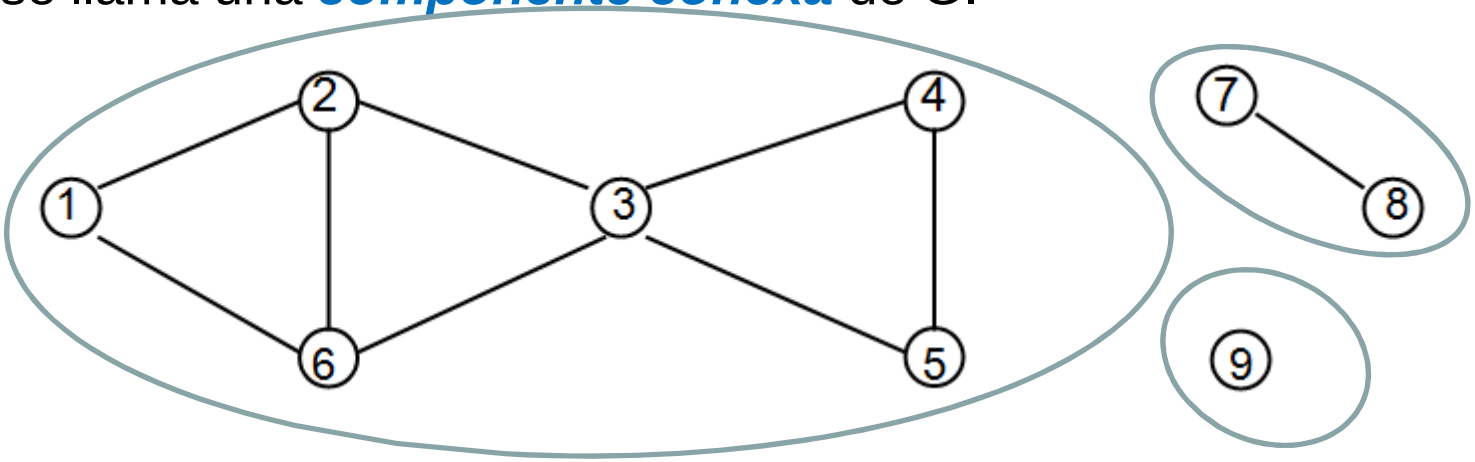
Un grafo es ***conexo*** si cualquier vértice de G es ***alcanzable*** desde cualquier otro vértice. Un vértice *v* es alcanzable desde *w* si hay una cadena que los une.

La relación ***es alcanzable desde*** es una relación de equivalencia en el conjunto V (tiene las propiedades reflexiva, simétrica y transitiva).

Por lo tanto como toda relación de equivalencia, induce en el conjunto V una partición en *clases de equivalencia*.

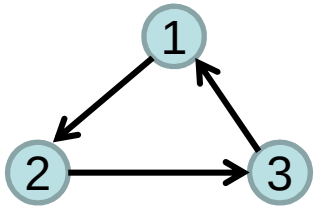
Cada subgrafo de G generado por una de tales clases de equivalencia se llama una ***componente conexa*** de G.

Ej.



Digrafo Fuertemente Conexo

Un **digrafo es *fuertemente conexo*** si dados dos vértices cualesquiera del grafo, existe un camino que parte de uno de ellos y llega al otro. En otras palabras si para todo par de vértices existe un circuito que los contiene.



Un **digrafo es *acíclico*** si no admite ningún ciclo. Un digrafo acíclico se denota también con ***dag*.**

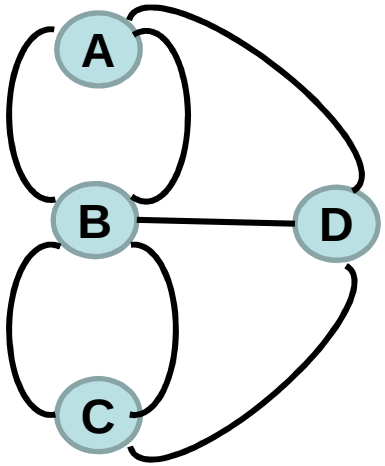
Un ***árbol*** es un grafo conexo y acíclico.

Un ***bosque*** es un grafo no conexo y acíclico.

Ciclo euleriano

Un **ciclo euleriano** es un ciclo que contiene todas las aristas del grafo en consideración.

En el ejemplo de los puentes de Koenigsberg, el problema que se formula es si el grafo en cuestión admite un ciclo euleriano.

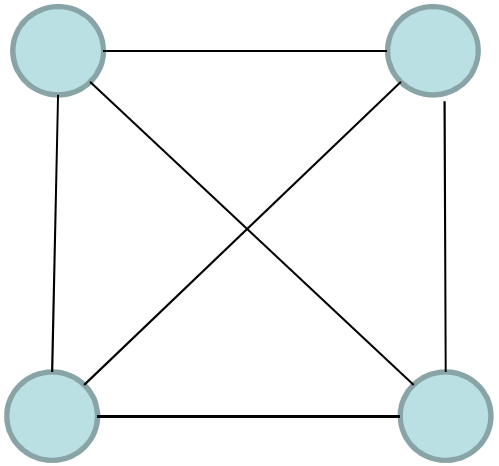


Teorema de Euler: un grafo conexo G admite un ciclo euleriano si y solo si todo vértice de G tiene grado par.

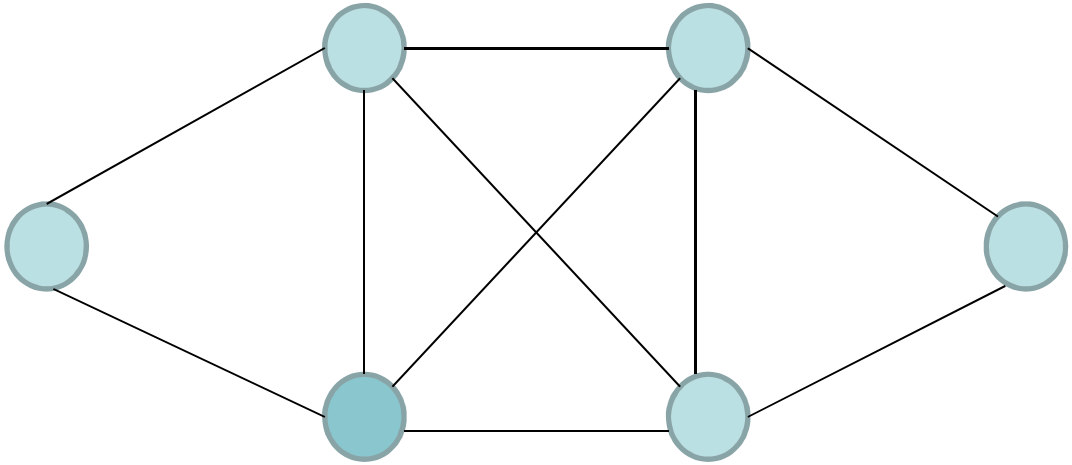
El teorema de Euler da un criterio de determinación del un ciclo euleriano. Pero no da el algoritmo para determinarlo en el caso de que exista.

Ciclo euleriano
Ejemplos

Ej 1. No admite ciclo euleriano:

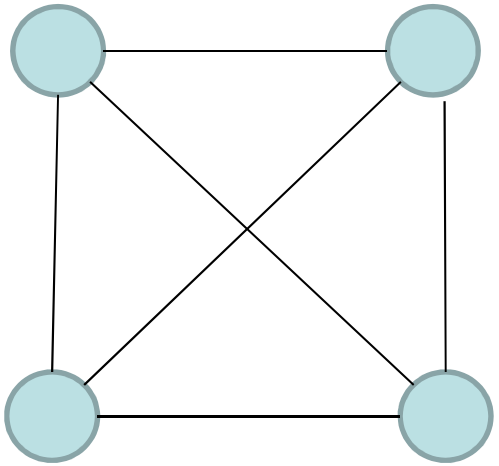


Ej 2. Si admite ciclo euleriano:

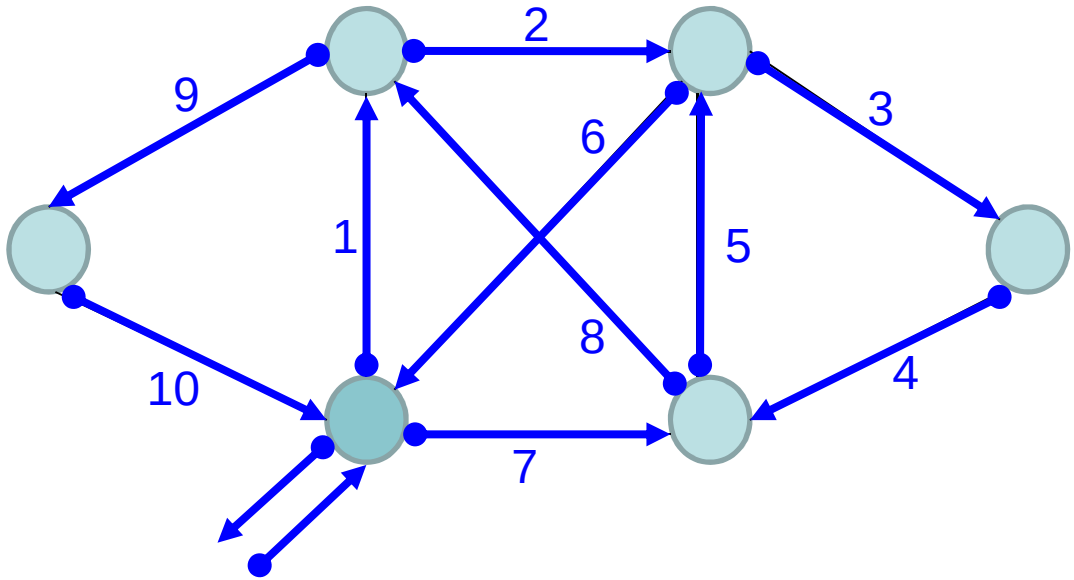


Ciclo euleriano
Ejemplos

Ej 1. No admite ciclo euleriano:



Ej 2. Si admite ciclo euleriano:

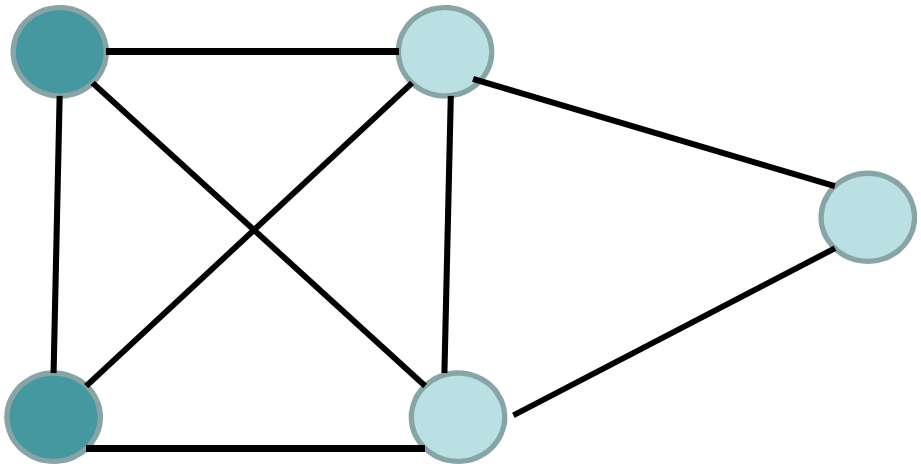


El algoritmo para encontrar un ciclo euleriano en un grafo se puede realizar en tiempo lineal O(a+n).

Camino euleriano

DEFINICION: Un grafo G admite una cadena euleriana si tiene como máximo dos vértices de grado impar.

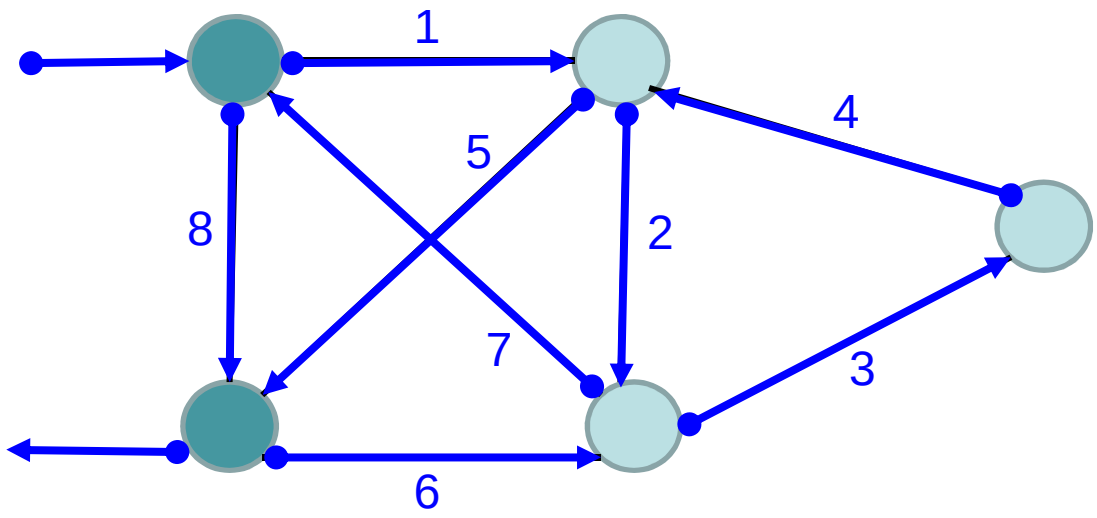
Ej 3. Admite camino euleriano:



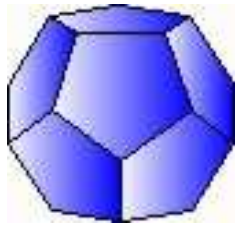
Camino euleriano

DEFINICION: Un grafo G admite una cadena euleriana si tiene como máximo dos vértices de grado impar.

Ej 3. Admite camino euleriano:



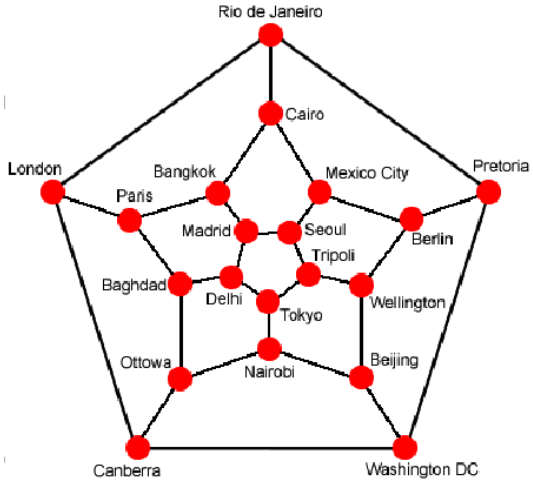
Ciclo hamiltoniano
Definición

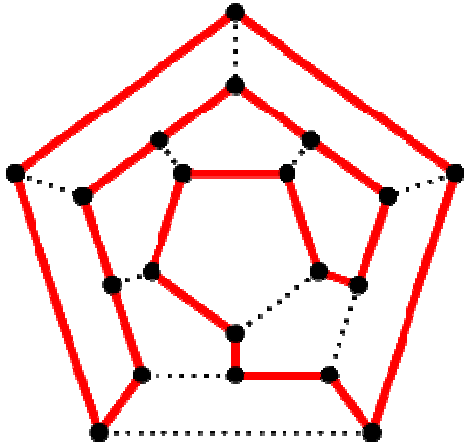


Un **ciclo hamiltoniano** es un ciclo simple (no repite vértices) que recorre todos los vértices del grafo.

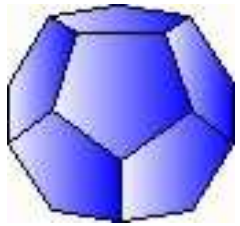
El nombre de " ciclo hamiltoniano " proviene de un juego de ingenio conocido como " La Vuelta al Mundo " y producido comercialmente a mediados del siglo XIX por Sir William R. Hamilton (1805-1865), famoso fisicomatemático irlandés.

El juego consistía básicamente en la determinación de un viaje cerrado que incluía todas las ciudades una sola vez, en un dodecaedro regular (12 caras), cuyos vértices (20) representaban ciudades importantes del mundo y sus aristas (30) eran las rutas que comunicaban dichas ciudades.



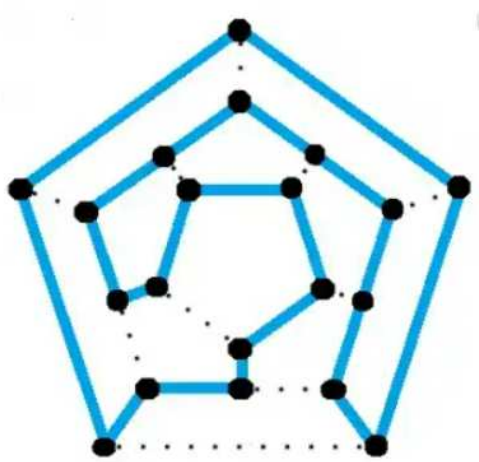


Ciclo hamiltoniano



El problema de determinar un ciclo Hamiltoniano tiene 2 partes:

- Formulación de un criterio que permita saber si existe o no un ciclo Hamiltoniano. No se conocen condiciones necesarias y suficientes para la existencia de circuitos hamiltonianos. Se conocen teoremas que dan condiciones suficientes para la existencia de circuitos hamiltonianos.



- Formulación de un algoritmo eficiente que si existe determine los ciclos Hamiltoniano**. Este es un problema NP completo.**

Problema del camino mínimo
Definición

Dado un digrafo G(V,E), en el cual los arcos tienen costo no negativo y dado un vértice origen v_0, el problema es ***determinar el camino de costo mínimo*** desde v_0 a cada uno de los vértices restantes del grafo.

En un grafo ponderado, el ***costo del camino*** es la suma de los costos de los arcos que lo forman.

Este problema puede resolverse por un algoritmo Greedy llamado ***ALGORITMO DE DIJKSTRA*** (1956).

El algoritmo obtiene siempre una solución óptima (si es que existe) y fue uno de los primeros algoritmos Greedy.

Algoritmo de Dijkstra

ALGORITMO DIJKSTRA (A,n,D,P)

Calcula el costo del camino de menor costo desde el vértice 1 a cada vértice del grafo.

Con recuperación de camino

ENTRADA: n: número de vértices.
A: matriz de adyacencia con los costos positivos.

SALIDA: D: vector de distancias especiales.
P: vector del camino.

AUXILIARES: S: conjunto de vértices elegidos.
C: conjunto de vértices candidatos

Algoritmo de Dijkstra

ALGORITMO DIJKSTRA (A,n,D,P)

P1. S ← { 1 } C ← { 2, 3, ..., n } P(1) ← 0

P2. Para i desde 2 hasta n hacer

D(i) ← A (1, i)

P(i) ← 1

P3. Repetir n-2 veces

Elegir en C un vértice w tal que D (w) sea mínimo

Agregar w a S

Borrar w de C

Para cada vértice v en C hacer

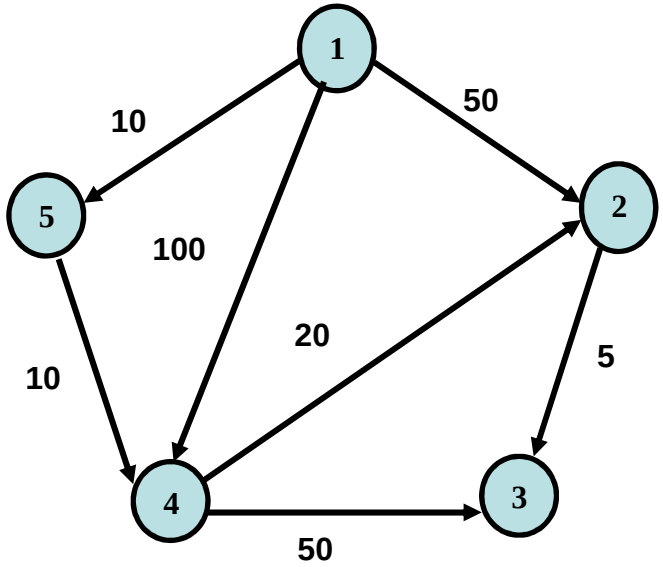
Si (D (w) + A (w , v)) < D (v) entonces

D (v) ← D (w) + A (w , v)

P (v) ← w

P4. Fin.

Algoritmo de Dijkstra
Ejemplo



$$A = \begin{pmatrix} 0 & 50 & \infty & 100 & 10 \\ \infty & 0 & 5 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 20 & 50 & 0 & \infty \\ \infty & \infty & \infty & 10 & 0 \end{pmatrix}$$

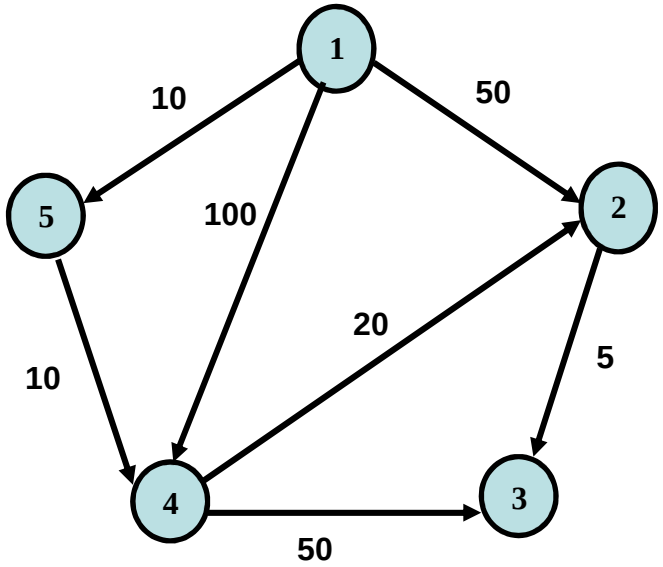
Inicial: S={1} C={2,3,4,5} D=(0 50 ∞ 100 10) P=(0 1 1 1 1)
it 1: w=**5** S={1,**5**} C={2,3,4} D=(0 50 ∞ **20** **10**) P=(0 1 1 **5** 1)
it 2: w=**4** S={1,5,**4**} C={2,3} D=(0 **40** **70** **20** 10) P=(0 **4** **4** 5 1)
it 3: w=**2** S={1,5,4,**2**} C={3} D=(0 **40** **45** 20 10) P=(0 4 **2** 5 1)

Resultados: **D=(0 40 45 20 10) P=(0 4 2 5 1)**

Algoritmo de Dijkstra
Ejemplo

Resultados del algoritmo de Dijkstra:

D=(0 40 45 20 10) **P=(0 4 2 5 1)**



Con el vector P se puede recuperar cuales son los vértices del camino de mínimo costo:

El camino de menor costo a cada vértice:

De 1 a 2 : P(2) = 4 , P(4)=5, P(5) = 1 , el camino es: 1→5→4→2

De 1 a 3 : P(3) = 2, P(2) = 4 , P(4)=5, P(5) = 1, el camino es: 1→5→4→2→3

De 1 a 4 : P(4)=5, P(5) = 1 , el camino es: 1→5→4

De 1 a 5 : P(5) = 1 , el camino es: 1→5

Algoritmo de Dijkstra

Dado un digrafo G con n nodos y con a arcos, el costo del algoritmo de Dijkstra depende de la implementación del grafo:

- Matriz de adyacencia:

$$\text{Algoritmo de Dijkstra } \epsilon\ O(n^2)$$

- Con una cola de prioridad implementada con heap:

$$\text{Algoritmo de Dijkstra } \epsilon\ O((a+n)\log n)$$

Problema de todos los Caminos Mínimos Definición

Dado un digrafo G=(V,E) en el que los arcos tienen costo no negativo, almacenados en la matriz C de adyacencia.

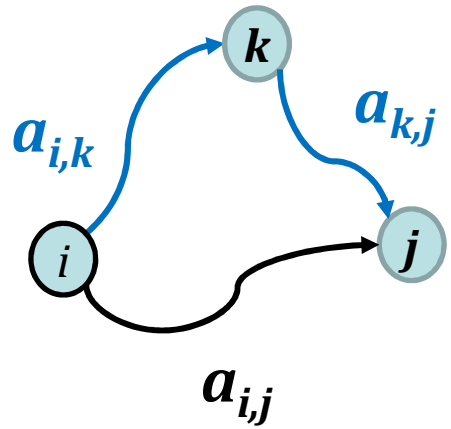
El problema consiste en ***encontrar el camino de menor costo para cada par ordenado de vértices*.**

Una solución es aplicar el algoritmo Greedy de Dijkstra para cada vértice tomándolo como origen.

Otra solución es usar una técnica de programación dinámica. En el algoritmo llamado de **Floyd** se seleccionan los vértices en orden secuencial para construir una matriz A de costos de caminos mínimos paso a paso.

Algoritmo de Floyd

Se define $a_{i,j}^{k}$ como la distancia del camino mas corto desde i a j que usa solo vértices $1..k$ como vértices intermedios.



Después de la k-ésima iteración $a_{i,j}^{k}$ contendrá los valores del camino mas corto desde i a j que pasa por vértices $1..k$ como vértices intermedios.

Después de la n iteraciones $a_{i,j}^{n}$ contendrá los valores del camino más corto desde i a j que pasa por todos los vértices del grafo

Algoritmo de Floyd

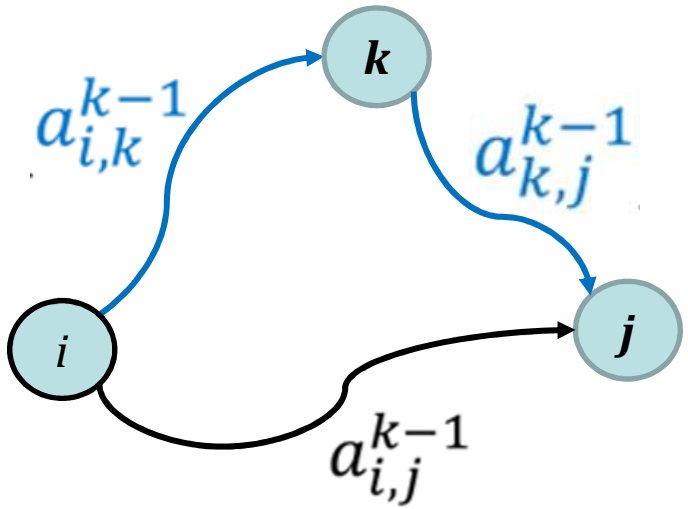
Por esta definición entonces:

$$a_{i,j}{}^{0} = c_{i,j} \qquad \forall i \;\; \forall j$$

$$a_{i,i}{}^{0} = 0 \qquad i=1..n$$

Para $k=1,2,..n$ hacer:

$$a_{i,j}^{k} = minimo \left\{ \begin{matrix} a_{i,j}^{k-1} \\ a_{i,k}^{k-1} + a_{k,j}^{k-1} \end{matrix} \right\}$$



Algoritmo de Floyd

ALGORITMO FLOYD (C,n,A)

Encuentra el camino de costo mínimo entre cualquier par de vértices del grafo.

ENTRADA:

n: número de vértices.

C: matriz de costos no negativos.

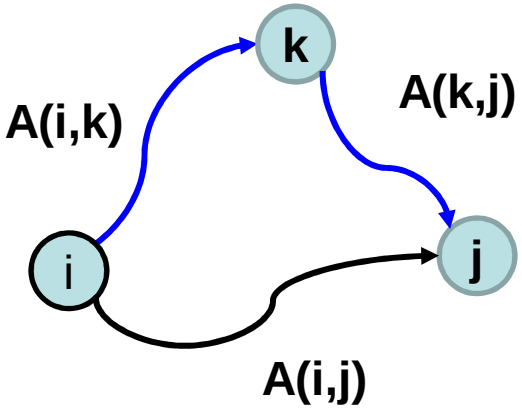
SALIDA:

A: matriz de costos de los caminos mínimos

Algoritmo de Floyd

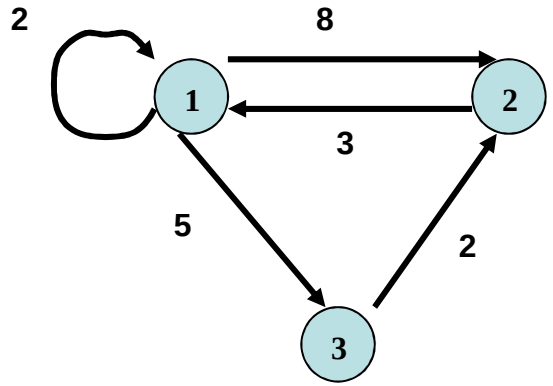
ALGORITMO FLOYD (C,n,A)

P1. Para i =1 hasta n hacer
    Para j =1 hasta n hacer
        A(i,j) ← C(i,j)
P2. Para i =1 hasta n hacer
    A(i,i) ← 0
P3. Para k =1 hasta n hacer
    Para i =1 hasta n hacer
        Para j =1 hasta n hacer
            Si (A(i,k) + A(k,j)) < A(i,j) entonces
                A(i,j) ← A(i,k) + A(k,j)
P4. Fin.



ALGORITMO FLOYD **ϵ $O(n^3)$**

Algoritmo de Floyd
Ejemplo



$$Costo = \begin{pmatrix} 2 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{pmatrix}$$

$$A^0 = \begin{pmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{pmatrix}$$

$$A^1 = \begin{pmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{pmatrix}$$

$$A^3 = \begin{pmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{pmatrix}$$

Clausura Transitiva
Definición

Un problema relacionado con el problema de encontrar todos los caminos mínimos en un grafo, consiste en *determinar si existe o no camino entre dos vértices del grafo*. Esto es, dados dos vertices i y j determinar si existe algún camino que conecte i con j (no importa la longitud ni el costo del camino)

Se puede modificar el algoritmo de Floyd para este problemas y es lo que se conoce como ALGORITMO DE WARSHALL (1962)

Si A es la matriz de adyacencia de G, entonces la **matriz A^+** se llama la ***matriz de clausura transitiva del grafo*** y tiene la propiedad que:

$a_{i,j}^+ = 1$ si existe un camino de longitud ≥1 desde i a j

$a_{i,j}^+ = 0$ si NO existe un camino de longitud ≥1 desde i a j

Algoritmo de Warshall

ALGORITMO WARSHALL (A,n,A^+)

*Calcula **A**$^+$ la clausura transitiva de la matriz **A** de adyacencia del grafo*

ENTRADA: n: número de vértices

A: matriz de adyacencia booleana

SALIDA: A^+: clausura transitiva del grafo

P1. $A^+ \leftarrow A$

P2. Para k desde 1 hasta n hacer

Para i desde 1 hasta n hacer

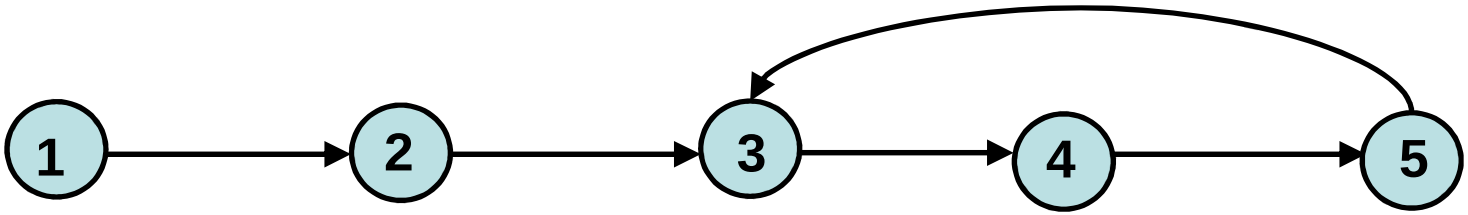
Para j desde 1 hasta n hacer

$A^+(i,j) \leftarrow A^+(i,j)$ or ($A^+(i,k)$ and $A^+(k,j)$)

P3. Fin.

ALGORITMO WARSHALL **ϵ $O(n^3)$**

Algoritmo de Warshall
Ejemplo



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad A^{+} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Orden Topológico
Definición

Dado un ***digrafo acíclico (dag)***, un orden topológico consiste en ordenar sus nodos de manera que si existe un camino del vértice *v* al vértice *w*, entonces *v* aparece antes de *w* en la clasificación.

Está claro que un orden topológico no es posible si el grafo tiene un ciclo, pues si dos vértices *v* y *w* están en un ciclo, *v* precede a *w* y *w* precede a *v*.

Más aun, **el orden topológico de un mismo grafo no es necesariamente único.**

Orden Topológico
Aplicación

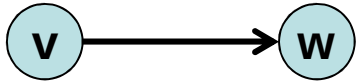
- Los ***digrafos acíclicos*** sirven para el planeamiento real de algunas actividades o problemas concretos.

- Al representar el planeamiento como un grafo, los nodos son las diferentes etapas del proyecto desde la etapa inicial hasta la etapa final, y los arcos son las actividades que tienen que se deben realizar para pasar de una etapa a otra.

- Si se considera que en este esquema de trabajo el que realiza la tarea es **una sola persona**, se trata de obtener un orden válido para la ejecución de las distintas subtareas que componen el proyecto.

Orden Topológico
Aplicación

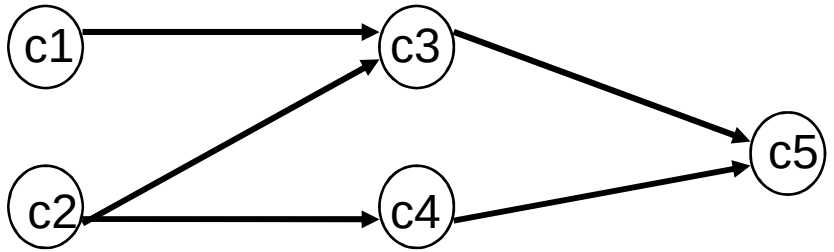
- Si se considera las materias correlativas de una carrera , los nodos corresponden a los cursos y los arcos dirigidos a las correlativas.
- Un arco (v,w) indica que el curso v es un requisito para cursar w y debe completarse antes de cursar w.



- Suponiendo que se toma **un curso a la vez**, un orden topológico de los cursos será una posible secuencia que no viola la estructura de los prerrequisitos.

Orden Topológico
Ejemplo

Asignatura	**Requisito**
c1	sin requisito
c2	sin requisito
c3	c1,c2
c4	c2
c5	c3,c4



Un digrafo que modele la situación tiene las asignaturas en sus nodos y los requisitos como arcos dirigidos entre los nodos como los requisitos.

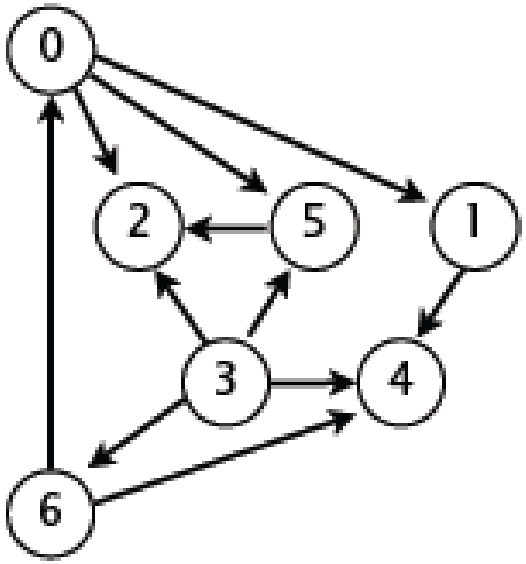
Un orden topológico:

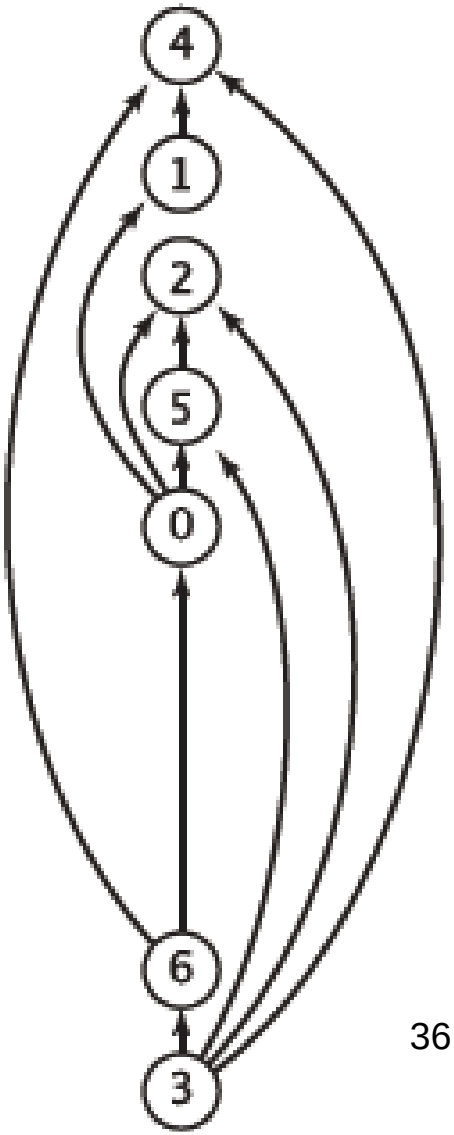
c1 c2 c3 c4 c5 o también: **c2 c4 c1 c3 c5**

Orden Topológico
Ejemplo

Dag G:

0→1,2,5
1→4
2→
3→2,4,5,6
4→
5→2
6→0,4





TS(G)=(3,6,0,5,2,1,4)