Estructura de Datos II

David Concha Gómez

Asignatura obligatoria

Segundo cuatrimestre

Créditos: 6

Moodle de la asignatura

Guía docente

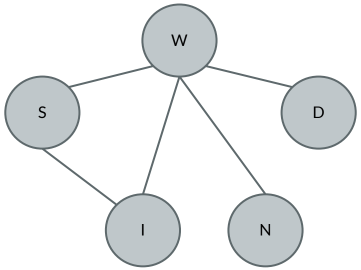
Índice

- Introducción a los grafos
- Matriz de adyacencia
- Lista de adyacencia
    - Tablas hash
- Algoritmos sobre grafos
    - Recorridos
    - Otros

Introducción a los grafos

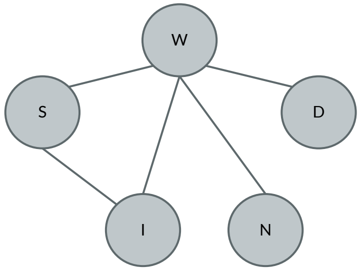
- Un grafo **G** es un par ordenado **G=(V, A)**, donde **V** es un conjunto de vértices o nodos y **A** es un conjunto de aristas o arcos.

- Aunque V puede ser infinito, nos limitaremos a grafos finitos.



Introducción a los grafos

- El siguiente grafo puede expresarse con los siguientes conjuntos:



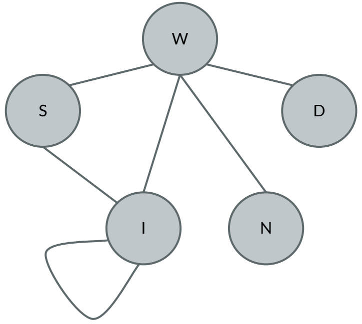
V = {W, S, D, I N}

A = {a1, a2, a3, a4, a5}

a1=(W, S)
a2 = (S, I)
a3 = (I, W)
a4 = (N, W)
a5 = (W, D)

Introducción a los grafos

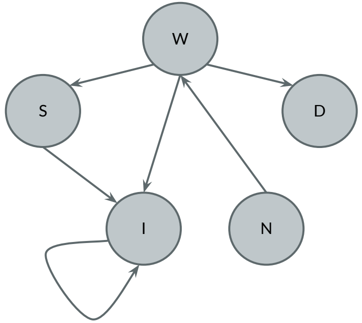
- Dos grafos **G=(V, A)** y G'=(V', A') se dicen isomorfos si existe una biyección f tal que **(u, v)∈A** si y solo si **(f(u), f(v))∈A'**.
- Un arista incide en un vértice si el vértice forma parte del par que define la arista.
- Dos aristas son adyacentes si tienen un vértice en común.
- Dos vértices son adyacentes o vecinos si existe una arista que los une.
- Un bucle es una arista que conecta un vértice consigo mismo.



Introducción a los grafos

- Si en un grafo las aristas no son simétricas, si existe la arista (W, S) no implica que exista la arista (S, W), se dice que el grafo es un grafo dirigido o digrafo.

- Una arista (W, S) se dice que W es su vértice inicial y S su vértice final.

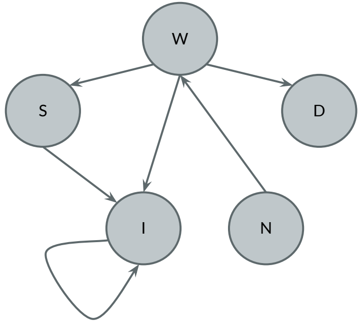


- Algunos autores proponen que los digrafos no pueden contener bucles, otros llaman digrafos simples a los digrafos sin bucles.

Introducción a los grafos

- En un digrafo se dice que una arista incide en un vértice si es el vértice final.

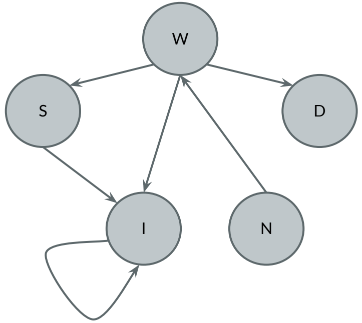
- Un nodo V_j es adyacente a otro nodo V_i si $(V_i, V_j) \in A$.



- Dos grafos son isomorfos bajo el mismo criterio. Esto implica que se respetan las direcciones de las aristas.

Introducción a los grafos

- Un camino es una secuencia ordenada de nodos adyacentes y las aristas que los unen. Si existe un camino entre dos nodos N y D, se dice que D es accesible por N.

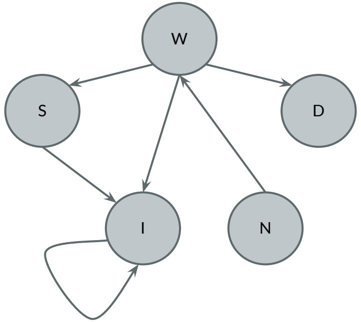


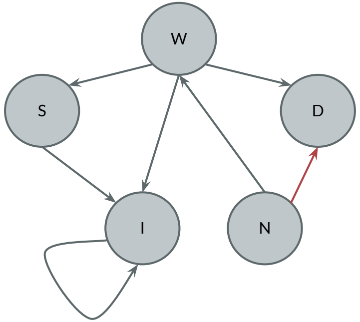
- Al número de aristas dentro de un camino se le denomina longitud.

- Un ciclo es un camino que empieza y termina en el mismo nodos sin repetir nodos intermedios.

Introducción a los grafos

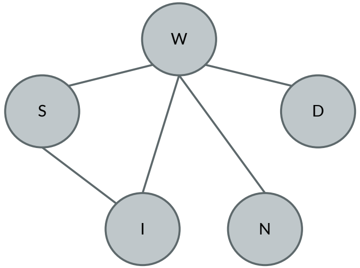
- La clausura o cierre transitivo de un grafo **G(V, A)** es otro grafo **G^+(V, A^+)** tal que para todo par (N_1, N_2) en **V** hay un arco en **A^+** si y solo si N_2 es accesible desde N_1.



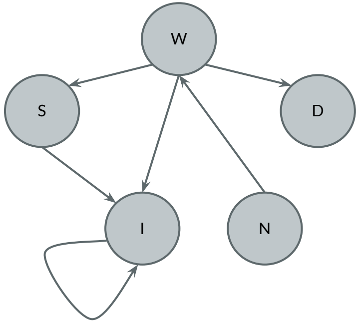


Introducción a los grafos

- En un grafo no dirigido, el grado de un vértice es igual al número de vértices adyacentes, contando los bucles como 2.

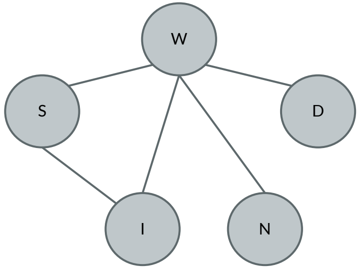


- En un grafo dirigido se diferencia entre:
  - grado de entrada: Número de aristas que tienen al vértice como final.
  - grado de salida: Número de aristas que tienen al vértice como inicial.

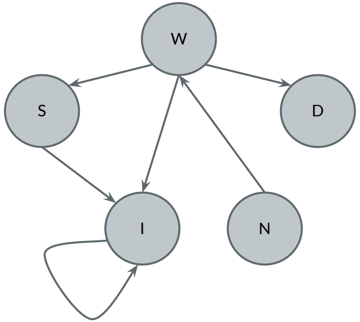


Introducción a los grafos

- En un grafo no dirigido se dice conexo si entre cada par de vértices existe un camino que los une.



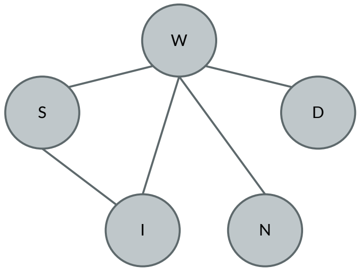
- Un grafo dirigido es fuertemente conexo si para cada par de vértices A y B, existe un camino de A hacia B y de B hacia A.

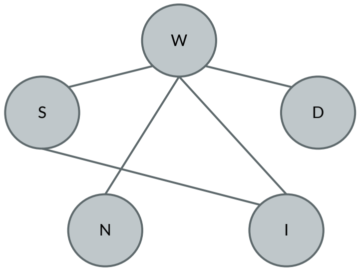


Introducción a los grafos

- En un grafo plano aquel que se puede dibujar en un plano cartesiano sin cruzar aristas.

- Un árbol es un grafo conexo sin ciclos.

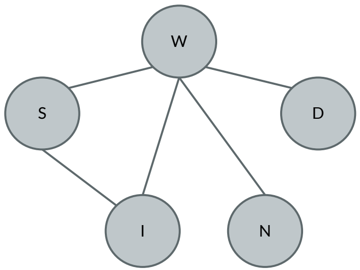


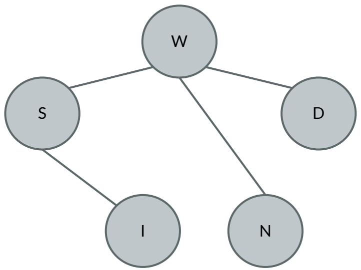


Introducción a los grafos

- Dado un grafo **G'=(V', A')** es un subgrafo de **G=(V, A)** si y solo si se verifica que:
    - A'⊆A
    - V'⊆V

- Un árbol se dice que es árbol de expansión o generador del grafo **G** si es un subgrafo de **G** que contiene todos los vértices de **G**.

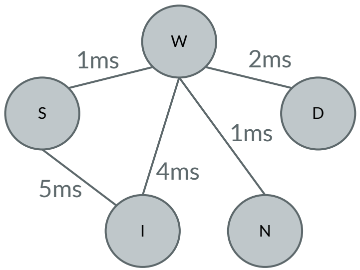


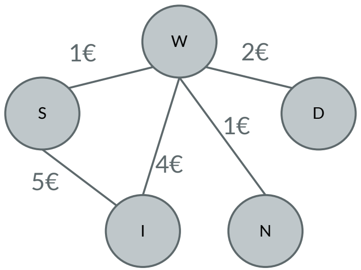


Introducción a los grafos

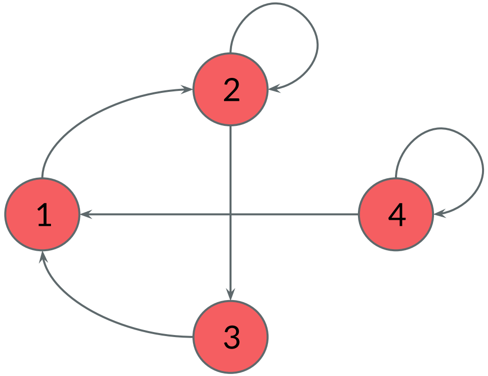
- Los grafos ponderados (valorados o con pesos) adjuntan a cada arista un valor (peso o coste)

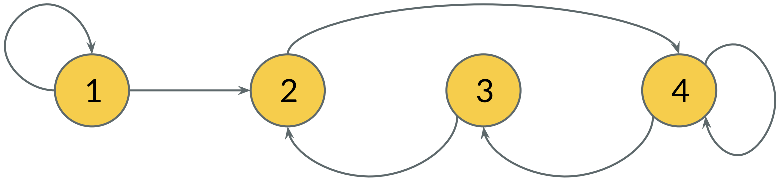
- El peso de una arista se suele ver como el coste asociado a transitar entre vértices vecinos.

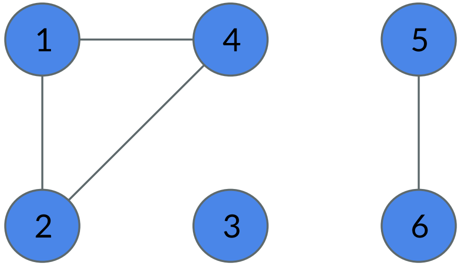


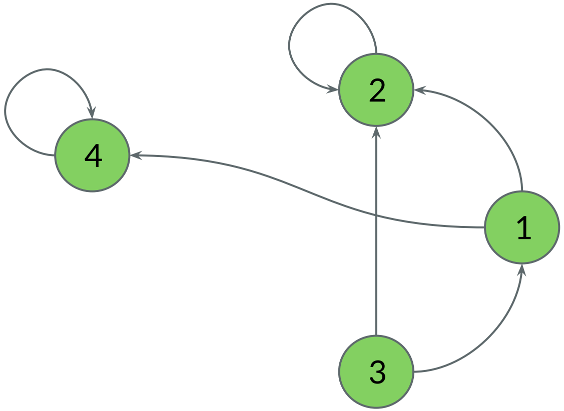


Introducción a los grafos

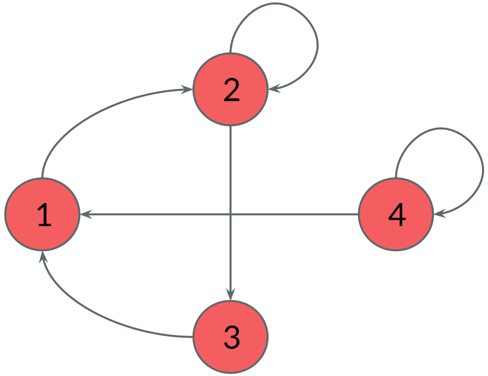


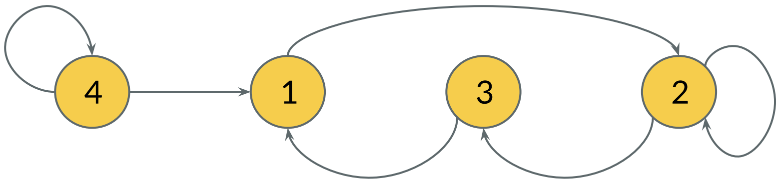


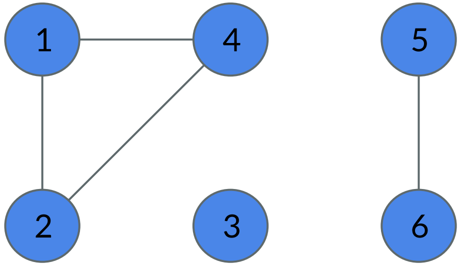


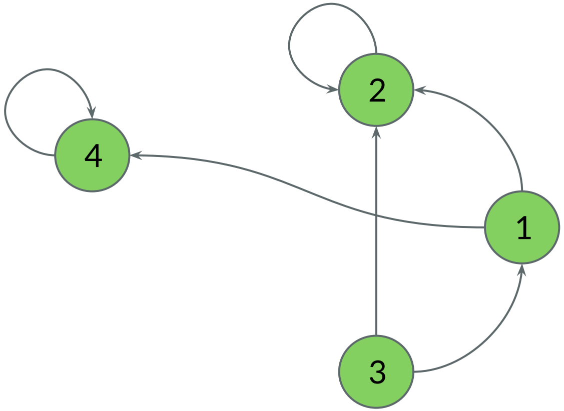


Introducción a los grafos





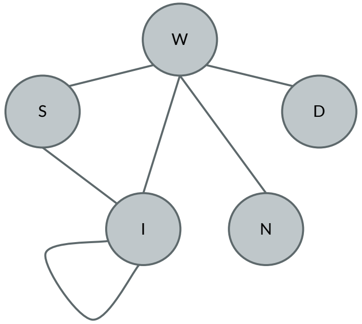




Matriz de adyacencia

- Es una matriz en la que filas y columnas representan a los nodos y cualquier elemento de la matriz a_{ij} representa el peso de los arcos cuyos extremos son (V_i, V_j).

- En los grafos no dirigidos la relación de adyacencia es simétrica

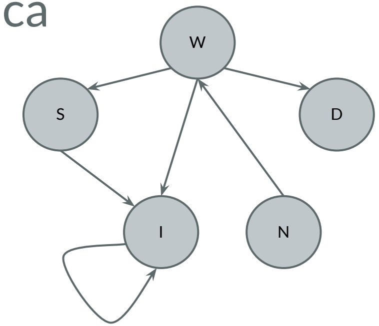


	W	S	D	I	N
W	0	1	1	1	1
S	1	0	0	1	0
D	1	0	0	0	0
I	1	1	0	1	0
N	1	0	0	0	0

Matriz de adyacencia

- Es una matriz en la que filas y columnas representan a los nodos y cualquier elemento de la matriz a_{ij} representa el peso de los arcos cuyos extremos son (V_i, V_j).

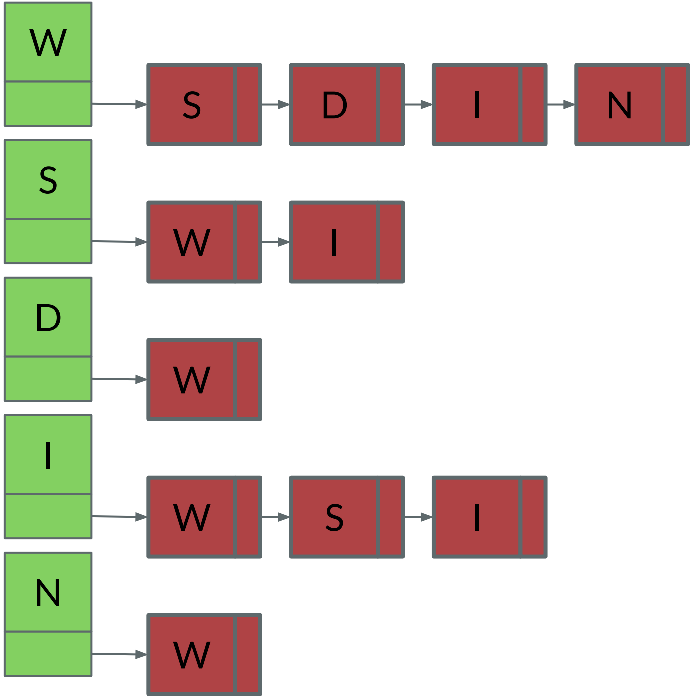
- En los digrafos la relación de adyacencia no tiene porque ser simétrica



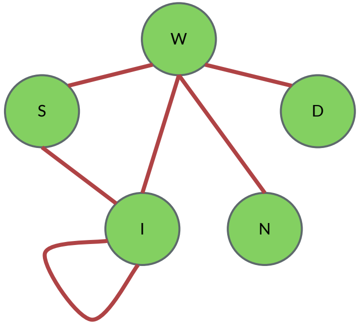
	W	S	D	I	N
W	0	1	1	1	0
S	0	0	0	1	0
D	0	0	0	0	0
I	0	0	0	1	0
N	1	0	0	0	0

Lista de adyacencia

- Un grafo es una lista de vértices en la que cada vértice se guarda una lista con sus aristas (de salida).



- Ideal cuando los grafos están lejos de ser completos.



Índice

- Introducción a los grafos
- Matriz de adyacencia
- Lista de adyacencia
    - Tablas hash
- Algoritmos sobre grafos
    - **Recorridos**
    - Otros

Recorrido en profundidad

- Permite visitar todo los nodos del grafo conexo.

- Procede profundizando por un camino hasta llegar a un punto en el que no es posible continuar, vuelve un paso atrás y explora otra alternativa.

- El recorrido obtenido no es homogéneo, caminos de la misma longitud no se exploran en instantes consecutivos.

Recorrido en profundidad

- Se necesita:
    - Una estructura para la **solución** (por ejemplo una lista).
    - Un **conjunto de visitados**.
    - Una **pila** para la exploración.
    - Inicialmente todas las estructuras están **vacías**.
- Algoritmo:
    - Se inserta el **nodo** inicial en la **pila**.
    - Mientras haya elementos en la **pila**:
        - Se extrae **la cima**.
        - Si ya está visitado, se descarta.
        - Si no está visitado:
            - Se marca como **visitado**.
            - Se añade a la **solución**.
            - Se añaden sus **adyacencias** a la **pila**.

Recorrido en anchura

- Permite visitar todo los nodos del grafo conexo.

- Se recorren todos los nodos adyacentes a uno dado antes de profundizar más en el grafo.

- El recorrido obtenido es homogéneo, ya que caminos de la misma longitud se exploran en instantes consecutivos.

Recorrido en profundidad

- Se necesita:
    - Una estructura para la **solución** (por ejemplo una lista).
    - Un **conjunto de visitados**.
    - Una **cola** para la exploración.
    - Inicialmente todas las estructuras están **vacías**.
- Algoritmo:
    - Se inserta el **nodo** inicial en la **cola**.
    - Mientras haya elementos en la **cola**:
        - Se extrae **el primero**.
        - Si ya está visitado, se descarta.
        - Si no está visitado:
            - Se marca como **visitado**.
            - Se añade a la **solución**.
            - Se añaden sus **adyacencias** a la **cola**.

Índice

- Introducción a los grafos
- Matriz de adyacencia
- Lista de adyacencia
  - Tablas hash
- Algoritmos sobre grafos
  - Recorridos
  - **Otros**

Camino más corto

- El recorrido en anchura permite obtener el camino más corto de un nodo al resto en grafos no ponderados.
- Para grafos ponderados se redefine como camino de menor coste.
- Dijkstra permite encontrar el camino de menor coste en grafos sin aristas negativas. Bellman-Ford es una solución más general para aristas negativas.

Camino más corto - Dijkstra

- Se necesita:
  - Una **tabla** con una fila por nodo. Por cada nodo:
    - **Distancia** para llegar a él (inicialmente infinito).
    - Un **conjunto de visitados** (inicialmente false).
    - Nodo **previo** (inicialmente nulo).
  - Una **cola de prioridad** de tuplas para la exploración.
  - Inicialmente todas las estructuras están **vacías**.

Nodo	Dist.	Visitado	Previo
A	∞	False	-
B	∞	False	-
…	∞	False	-

- Algoritmo:
  - Se inserta la **arista** inicial con coste cero y su nodo previo nulo en la **cola de prioridad**.
  - Mientras haya elementos en la **cola de prioridad**:
    - Se extrae **el primero**.
    - Se añade a la **solución** y se marca como **visitado**.
    - Por cada **arista**:
      - Si el **coste** de llegar al nodo **más** el de la **arista** es menor que el acumulado, se añade a la **cola de prioridad**.

Camino más corto - BellmanFord

- Se necesita:
  - Una **tabla** con una fila por nodo. Por cada nodo:
    - **Distancia** para llegar a él (inicialmente infinito).
    - Nodo **previo** (inicialmente nulo).
- Algoritmo:
  - Se marca el **nodo inicial** como **coste cero** y su **nodo previo nulo**.
  - **Repetir** N-1 veces:
    - Por cada **arista** del grafo:
      - La arista conecta *u* con *v*.
      - Si distancia[v] > distancia[u] + peso(u, v).
        - Se actualiza su **distancia** y se marca *u* como su **nodo previo**.
  - Por cada **arista** del grafo:
      - Si se sigue cumpliendo que distancia[v] > distancia[u] + peso(u, v):
        - **No hay solución**, hay ciclos negativos

Nodo	Dist.	Previo
A	∞	-
B	∞	-
…	∞	-

Árbol de expansión mínimo

- El árbol de expansión mínimo de un grafo con pesos, es un árbol que contiene todos los nodos del grafo y minimiza la suma de los pesos.

- Dos algoritmos voraces clásicos para resolver este problema son:
    - Kruskal
    - Prim

Árbol de expansión mínimo - Kruskal

- Se necesita:
  - Una estructura con las **aristas ordenadas por peso**.
  - Una estructura con un **conjunto** por **cada nodo**.
  - Una estructura para la **solución**.
- Algoritmo:
  - **Mientras** haya **aristas** o más de un **conjunto**:
    - Se elige la menor arista:
      - Si sus nodos están en conjuntos distintos:
        - Se añade la arista a la solución.
        - Se fusionan los conjuntos.
      - En caso contrario se descarta la arista.
  - Por **solución** es la colección de aristas

Árbol de expansión mínimo - Prim

- Se necesita:
  - Una **tabla** con una fila por nodo. Por cada nodo:
    - **Distancia** para llegar a él (inicialmente infinito).
    - Nodo **previo** (inicialmente nulo).
- Algoritmo:
  - Se marca el **nodo inicial** como **coste cero** y su **nodo previo nulo**.
  - **Repetir** N veces:
    - Por cada **arista** del grafo:
      - La arista conecta *u* con *v*.
      - Si distancia[v] > peso(u, v).
        - Se actualiza su **distancia** y se marca *u* como su **nodo previo**.
  - Por **solución** es la colección de aristas

Nodo	Dist.	Previo
A	∞	-
B	∞	-
…	∞	-

Cierre transitivo

- Se puede usar el recorrido en profundidad para resolverlo aplicando el recorrido desde cada uno de los nodos.

- En matrices de adyacencia su complejidad es de $O(n^3)$.

- Para listas de adyacencia su complejidad es de $O(n(n+m))$ que se aproxima a $O(n^3)$ para grafos densos.

Cierre transitivo

- Floyd-Warshall es un alternativa muy simple para obtener el cierre transitivo de un grafo.

- Busca iterativamente si un par de nodos están conectados por un nodo intermedio.

- La complejidad del algoritmo Floyd-Warshall también es $O(n^3)$.

Cierre transitivo - Floyd-Warshall

- Se necesita:
  - El grafo.
- Algoritmo:
  - Por cada **nodo** *k* del grafo:
    - Por cada **nodo** *u* del grafo:
      - Por cada **nodo** *v* del grafo:
        - Si *u* y *v* son distintos…
        - Si existe (*u*, *k*)...
        - Si existe (*k*, *v*)...
        - Si no existe (*u*, *v*)...
        - Añadir (*u*, *v*)

Estructura de Datos II

David Concha Gómez

Asignatura obligatoria

Segundo cuatrimestre

Créditos: 6

[Moodle de la asignatura]

[Guía docente]