



UNIVERSITAT DE BARCELONA



Estructura de datos

Grafos

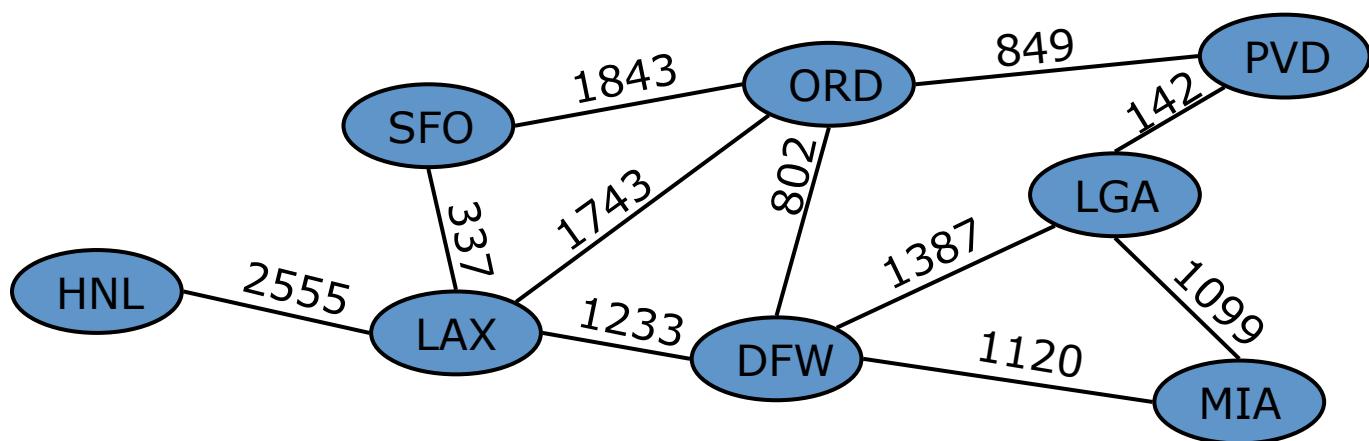
Santi Seguí | 2014-15

Índice

1. ¿Qué es un grafo?
2. Terminología
3. Propiedades de los Grafos
4. Tipos de Grafos
5. Representaciones
6. Performance
7. BFS/DFS
8. Aplicaciones

¿Qué es un grafo?

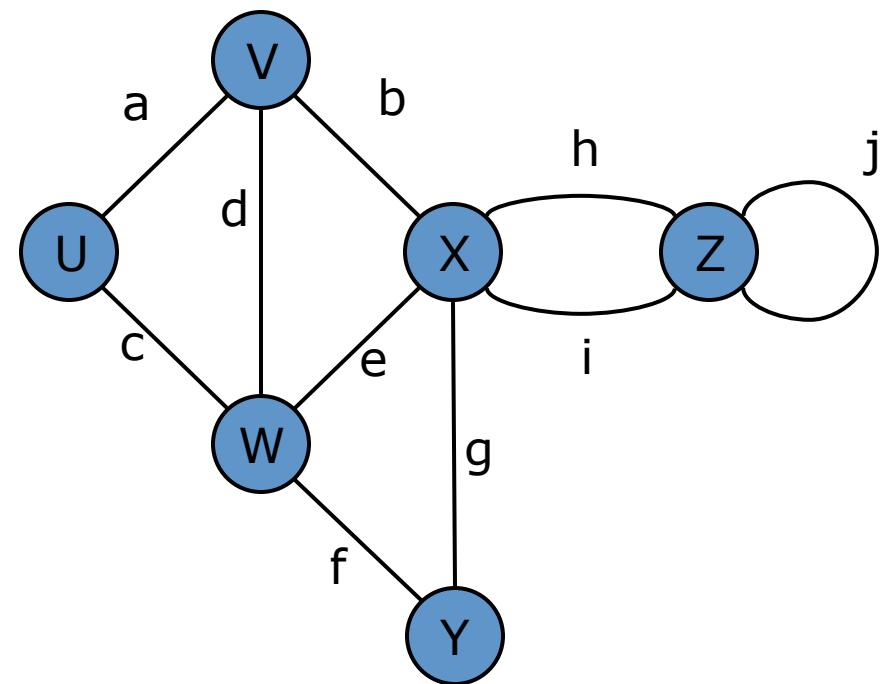
- Un grafo está definido por :
 - Un conjunto de vértices (V)
 - Un conjunto de aristas (E)
- Vértices y aristas pueden almacenar información



- Ejemplo:
 - Un vértice representa un aeropuerto y guarda las 3 letras correspondiente al código del aeropuerto
 - Una arista representa una ruta aérea entre dos aeropuertos y almacena el número de kilómetros de la ruta.

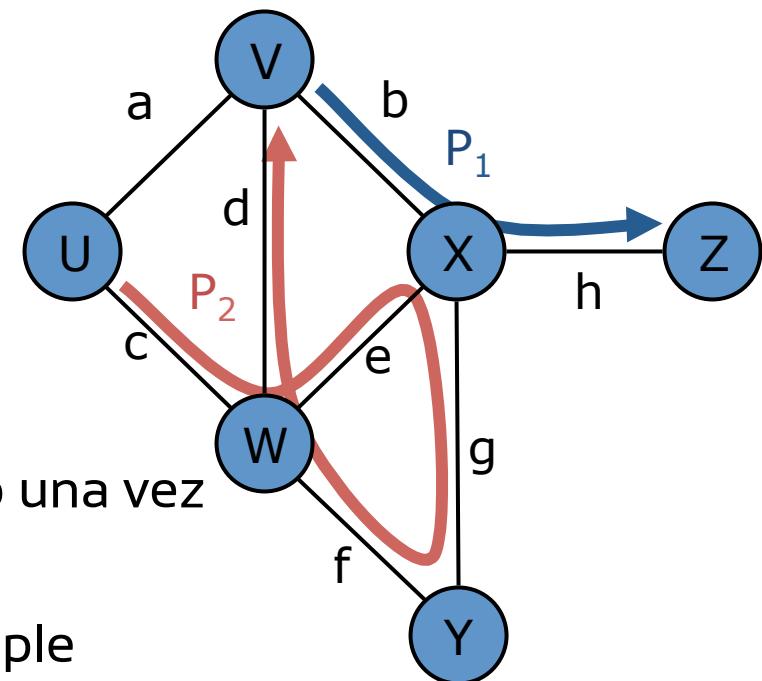
Terminología

- Puntos finales (o endpoints en inglés) de una arista
 - **U** y **V** son los puntos finales de **a**
- Aristas incidentes de un vértice
 - **a, d, y b** son incidentes en **V**
- Vértices adyacentes
 - **U** y **V** son adyacentes
- Grado de un vértice
 - **X** tiene grado 5
- Aristas paralelas
 - **h y i** son aristas paralelas
- Auto-ciclo (self-loop en inglés)
 - **j** es un auto-ciclo



Terminología (2)

- Camino
 - Secuencia de vértices y aristas alternativas
 - Empieza en un vértice
 - Termina en un vértice
 - Cada arista es precedida y seguida por sus puntos finales.
- Camino simple
 - Camino tal que todos sus vértices y aristas son visitadas como mucho una vez
- Ejemplo
 - $P_1 = (V-b->X-h->Z)$ es un camino simple
 - $P_2 = (U-c->W-e->X-g->Y-f->W-d->V)$ no es una camino simple pero si que es un camino.

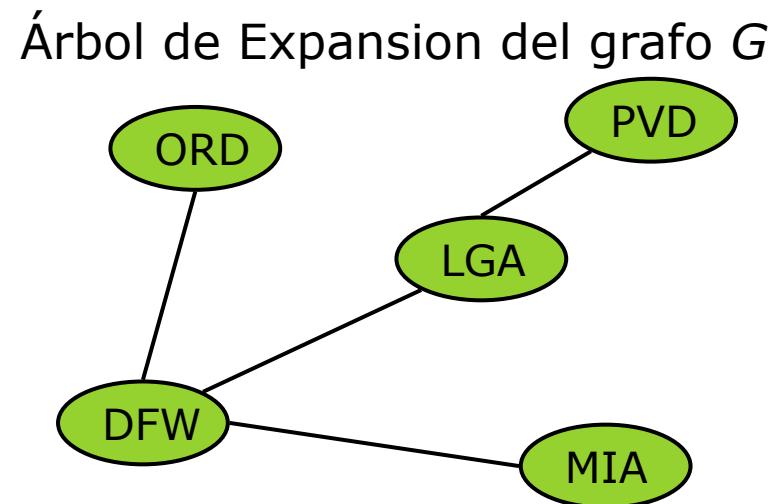
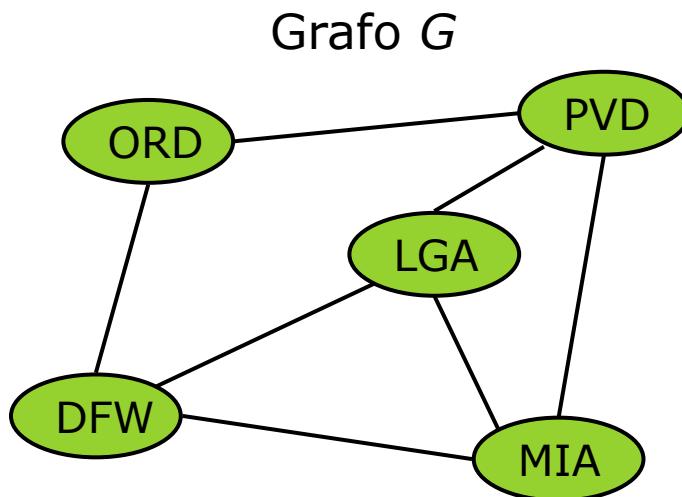


Propiedades de los Grafos

- Un grafo $G' = (V', E')$ es un subgrafo de $G = (V, E)$ si V' esta contenido en V y E' está contenido en E
- Un grafo esta **conectado** si existe un camino desde cada vértice a cualquier otro vértice del grafo G .
- Un **ciclo** es un camino que empieza y termina en el mismo vértice
- Un grafo es **acíclico** si ningún subgrafo tiene un ciclo

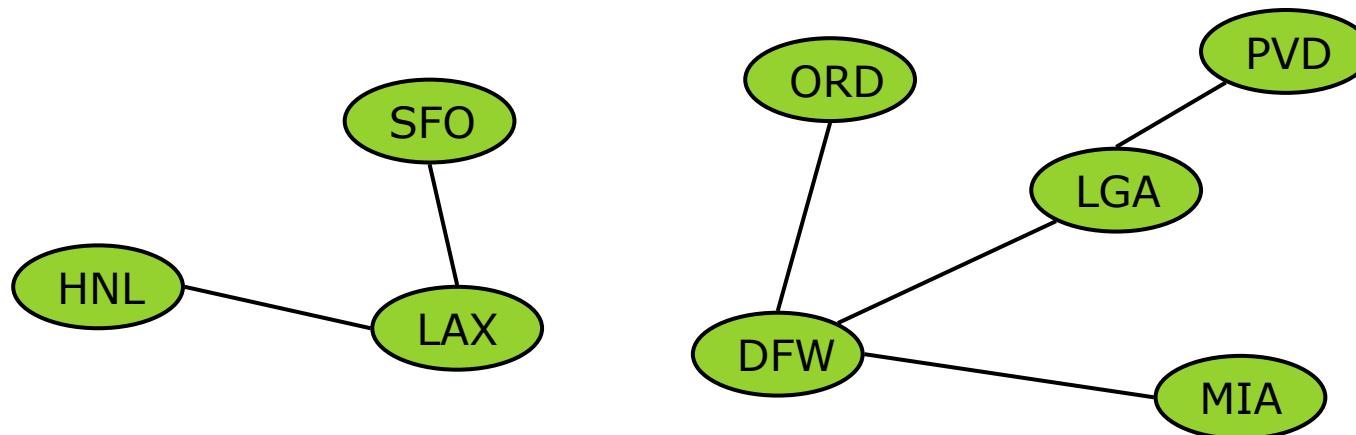
Propiedades de los Grafos (2)

- Un árbol de expansión (spanning tree en inglés) es un subgrafo que contiene todos los vértices del grafo en un árbol y suficientes aristas para conectar todos los vértices sin ningún ciclo



Propiedades de los Grafos (3)

- Un spanning forest es un subgrafo que consiste de un árbol de expansión (spanning tree) en cada componente conectado del grafo.
- Los Spanning forests nunca contienen ciclos.



Propiedades de los Grafos (4)

- Un **Grafo G** es un **árbol** solo si este satisface cualquier de las siguientes 5 condiciones:
 - G tiene $V-1$ aristas y no tiene ciclos
 - G tiene $V-1$ aristas y esta conectado
 - G esta conectado, pero eliminando cualquier de sus aristas el árbol deja de ser conectado
 - G es acíclico, pero añadiendo cualquier nueva arista se crea un ciclo
 - Solo un camino simple une toda pareja de vértices en el grafo G

Demostración 1

Demostremos que la suma del grado de todos los vértices del grafo G es igual al doble que el número de aristas del grafo G.

- Llamemos $V = \{v_1, v_2, \dots, v_p\}$, donde p es el número de vértices en G . Cuando calculamos la suma total de grados D , vemos que:

$$D = \deg(v_1) + \deg(v_2) + \dots + \deg(v_p).$$

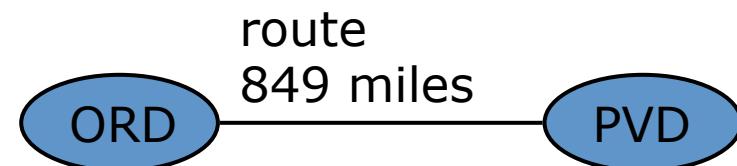
- Podemos ver que cada vértice lo contamos dos veces en D : una para cada uno de los dos vértices adyacentes de una arista. Por tanto $D = 2*|E|$, donde $|E|$ es el número de aristas.

Tipos de Grafos

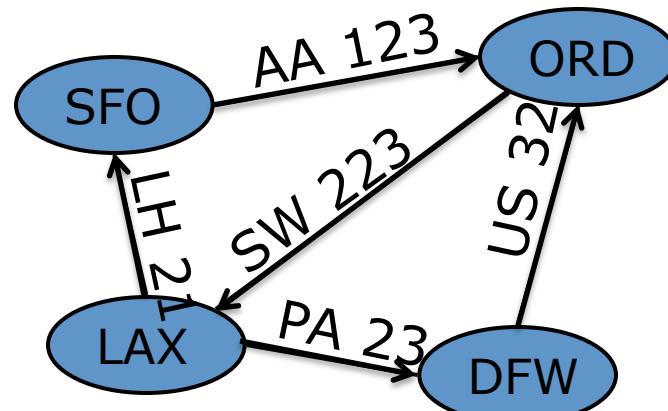
- Grafo dirigido
 - Todos sus aristas son dirigidas
 - Un arista es una pareja ordenada de vértices (u,v)
 - El primer vértice u en el origen
 - El segundo vértice v es la destinación
 - e.g., vuelos aéreos



- Grafo no dirigido
 - Todas sus aristas no son dirigidas
 - Una arista es un par de vértices sin orden (u,v)
 - e.g., ruta aérea

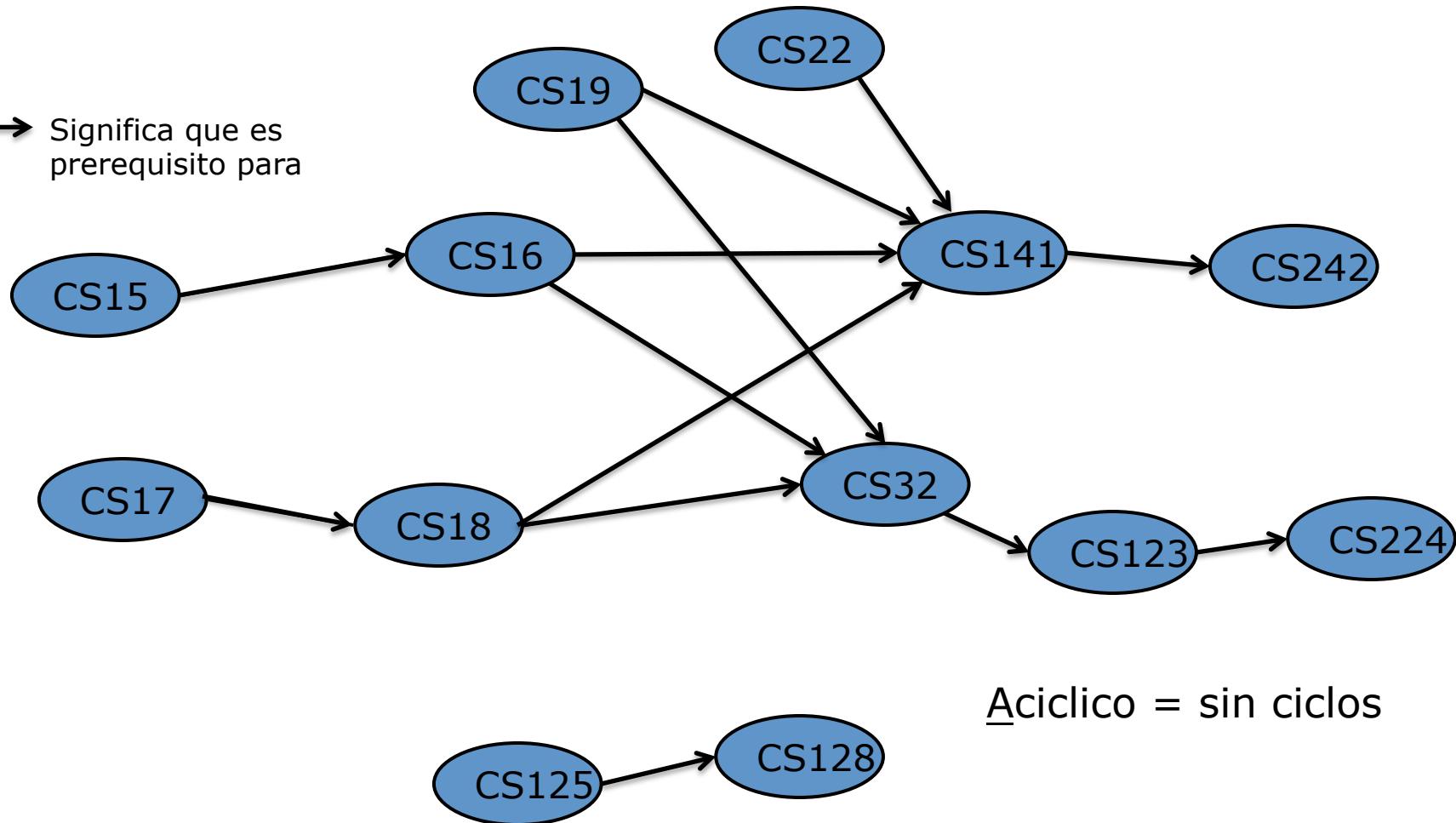


Tipos de Grafos: Grafo dirigido

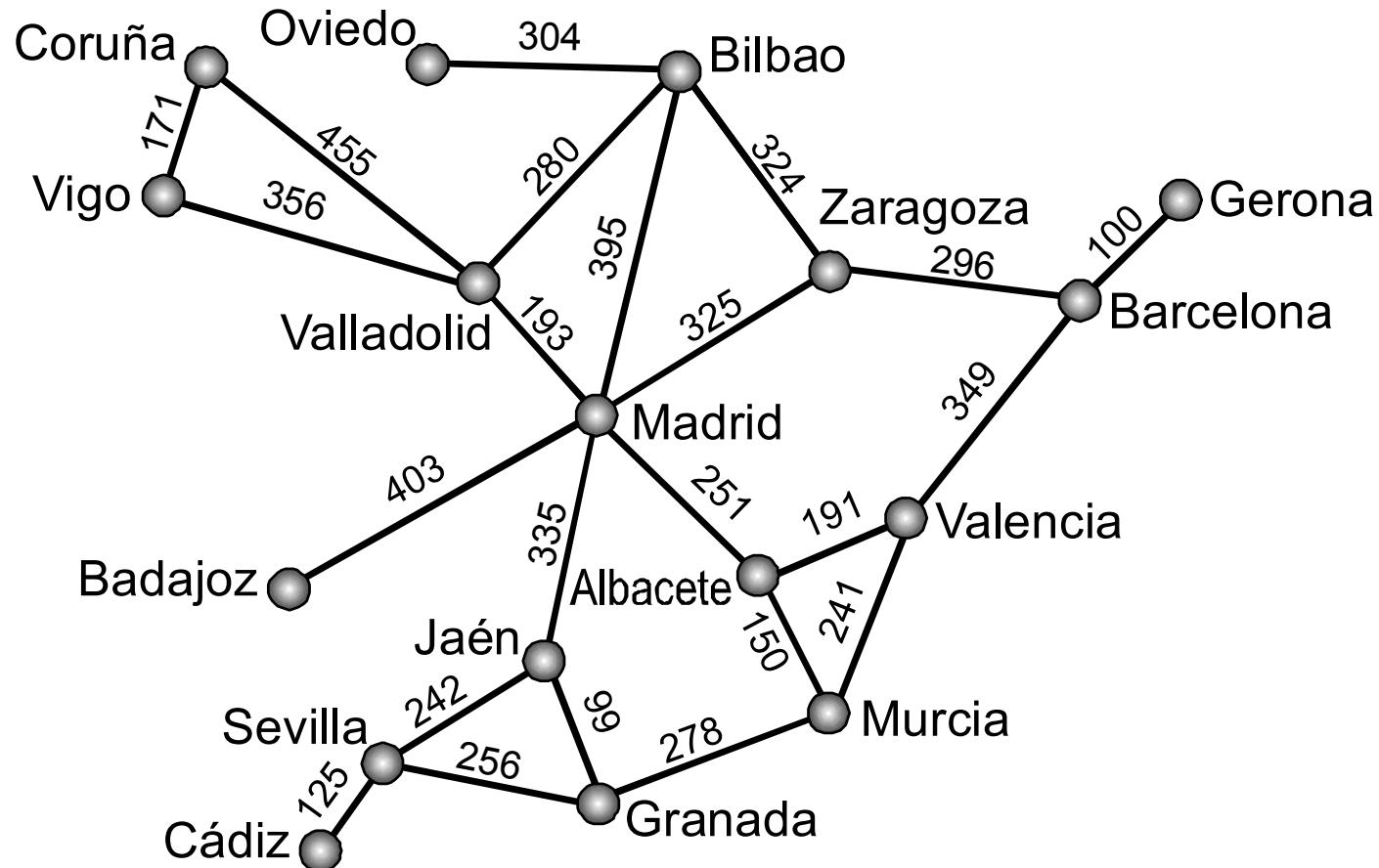


Grafo Acíclico dirigido (DAG)

→ Significa que es
prerequisito para



Grafo no dirigido



¿Qué problemas podemos definir sobre el modelo de carreteras entre ciudades?

¿Existe camino entre Cádiz y Barcelona?

¿Cuál es el camino más corto de Cádiz y Barcelona?

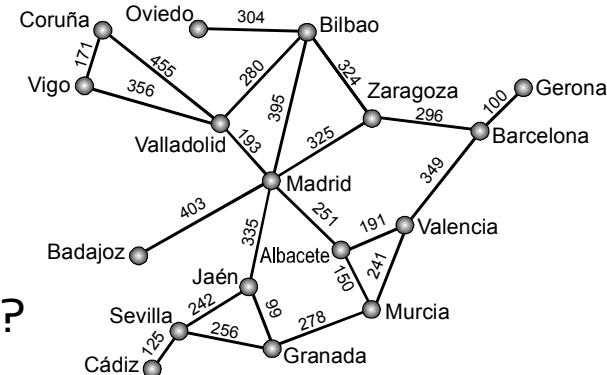
¿Existen algún camino que una dos ciudades cualquiera?

¿Cuál es la ciudad más lejana a Barcelona?

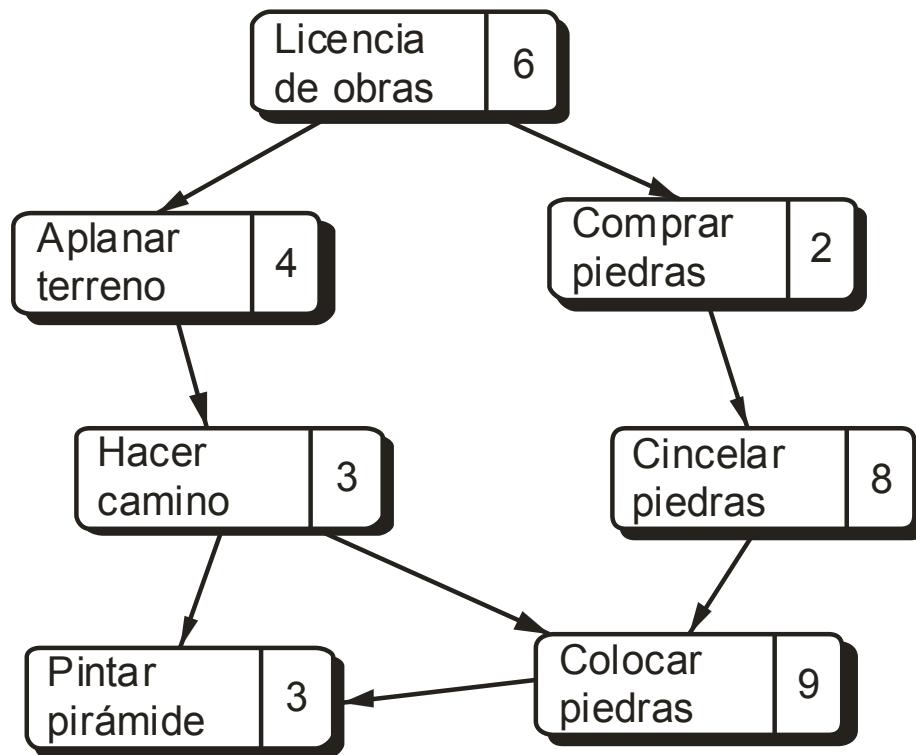
¿Cuál es la ciudad más céntrica?

¿Cuántos caminos distintos existen de Sevilla a Zaragoza?

¿Cómo hacer un tour pasando por todas las ciudades haciendo el menor número de kilómetros?



¿Cómo podemos modelar un mapa de planificación de tareas?



¿Cómo podemos modelar un mapa de planificación de tareas?

¿En cuánto tiempo, como mínimo, se puede construir la casa?

¿Cuándo debe empezar cada tarea en la planificación óptima?

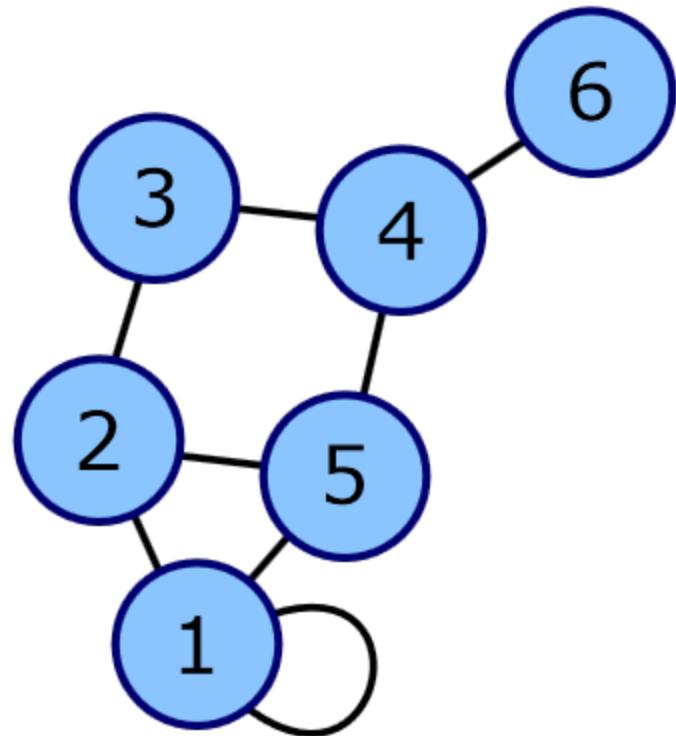
¿Qué tareas son más críticas (es decir, no pueden sufrir retrasos)?

¿Cuánta gente necesitamos para acabar las obras?

Representación de grafos

- Los vértices normalmente son guardados mediante listas o HashSet
- Hay tres métodos que son utilizados normalmente para guardar que vértices son adyacentes:
 - **Lista de aristas (o conjunto)**
 - **Listas de adyacencia (o conjuntos)**
 - **Matriz de Adyacencia**

Lista de aristas (o conjunto)

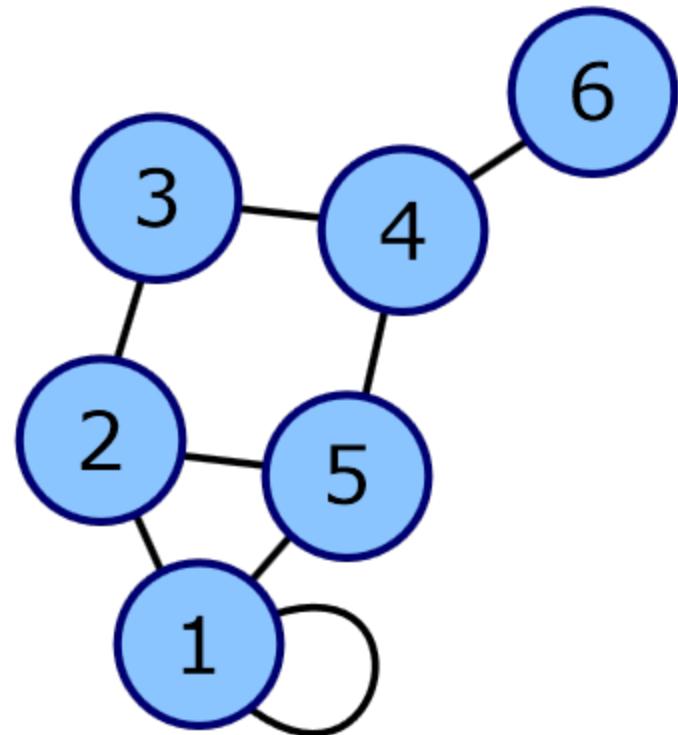


- Forma de representar que vértices son adyacentes a los otros mediante parejas de vértices
- Cada elemento en la lista es una arista (a,b) que representa la conexión de un nodo a a un nodo b

Lista de aristas:

[(1,1), (1,2), (1,5), (2,3), (2,5), (3,4), (4,5), (4,6)]

Lista de Adyacencia (o Conjuntos)



- Cada vértice tiene una lista que asocia los vértices vecinos

1	2	3	4	5	6
[1,2,5]	[1,3,5]	[2,4]	[3,5,6]	[1,2,4]	[4]

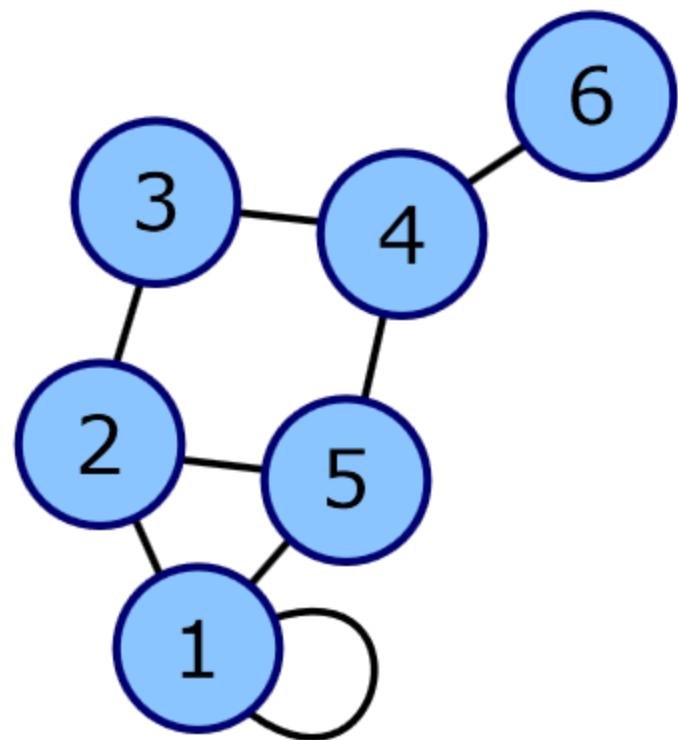
Conjuntos

- Recordemos que encontrar un elemento en un conjunto hash es una operación constante:
 $A = \{ (1,1), (1,2), (1,5), (2,3), (2,5), (3,4), (4,5), (4,6) \}$
 $A.contains((4,5))$ is $O(1)$
- La lista de adyacencia puede también estar representada por un hashsets:
 $B = [\{1, 2, 5\}, \{1, 3, 5\}, \{2, 4\}, \{3, 5, 6\}, \{1, 2, 4\}, \{4\}]$
 $B[2].contains(5)$ is still $O(1)$
- Hasta podríamos implementarlo mediante un hashset de hashsets!

Matriz de Adyacencia

- Otra manera de representar que vértices son adyacentes a otros
- Creamos una Matriz M de tamaño $n \times n$, donde n es el número de nodos del grafo
 - true = hay una arista
 - false = no hay arista
- Si $m[u][v]$ es true, entonces el nodo u tiene una arista que lo conecta con el nodo v (si el grafo es no dirigido, podemos asumir que también hay en la dirección opuesta)

Matriz de Adyacencia (2)



	1	2	3	4	5	6
1	T	T	F	F	T	F
2	T	F	T	F	T	F
3	F	T	F	T	F	F
4	F	F	T	F	T	T
5	T	T	F	T	F	F
6	F	F	F	T	F	F

Matriz de Adyacencia (3)

- Inicializamos una matriz de tamaño $n \times n$, donde n es el número de vértices que creemos que tendremos en nuestro grafo (si finalmente necesitamos mas vértices podemos expandir nuestra matriz fácilmente)
- Cuando se añade un vértice al grafo, tenemos que reservar una columna y fila para ese nuevo vértice
- Cuando se elimina un vértice, toda su columna y fila le asignaremos un valor false
 - Dado que no podemos eliminar filas/columnas de los arrays, mantendremos una lista de vértices que representan los vértices actuales en nuestro grafo.

Métodos principales de los Grafos ADT

- Vértices y Aristas
 - store values
 - Ex: edge weights
- Métodos de acceso
 - vertices()
 - edges()
 - incidentEdges(vertex)
 - areAdjacent(v1, v2)
 - endVertices(edge)
 - opposite(vertex, edge)
- Métodos de modificación
 - insertVertex(value)
 - insertEdge(v1, v2)
 - removeVertex(vertex)
 - removeEdge(edge)

Complejidad

	Edge Set	Adjacency Sets	Adjacency Matrix
Espacio necesario	$ V + E $	$ V + E $	$ V ^2$
vertices() ¹	1	1	1
edges()	1	$ E $	$ V ^2$
incidentEdges(v)	$ E $	1	$ V $
areAdjacent (v ₁ , v ₂)	1	1	1
insertVertex(val)	1	1	$ V $
insertEdge(v ₁ , v ₂)	1	1	1
removeVertex(v)	$ E $	$ V $	$ V $
removeEdge(v ₁ , v ₂)	1	1	1

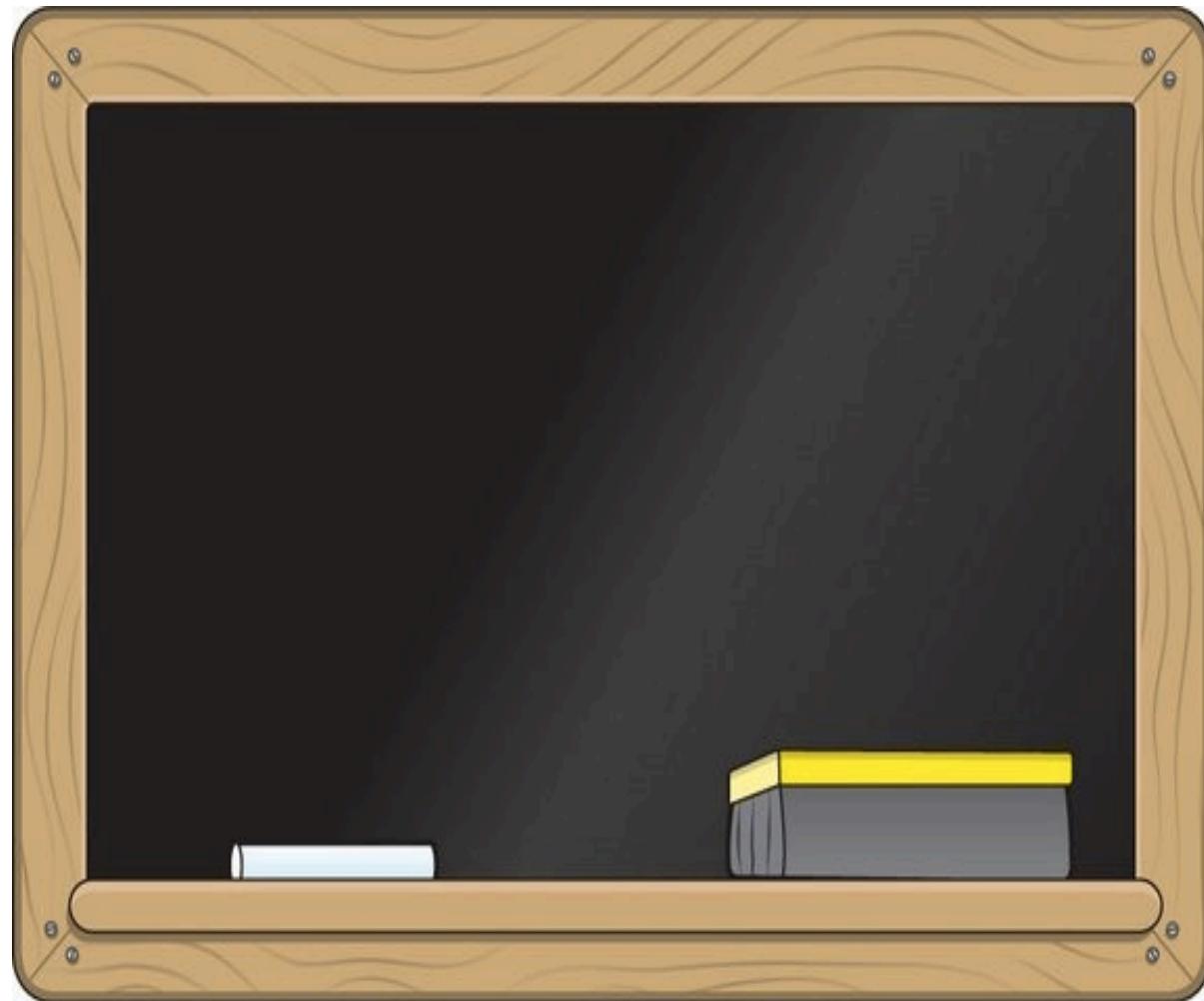
¹ En las tres implementaciones, mantenemos una lista adicional para los vértices

BFT/DFT

- ¿Recordáis como hacemos los recorridos por anchura/profundidad de los árboles?
- Los recorridos de los árboles también se pueden aplicar a los grafos!
 - (Un árbol es un tipo especial de grafo...)
 - Puede ser utilizado para **recorrer** y “visitar” los nodos del grafo (BFT / DFT)
 - Normalmente usados para **buscar** un valor/elemento concreto en el grafo (BFS / DFS)

Recorrido por anchura (BFT): Tree vs. Graph

```
function treeBFT(root):
    //Input: Root node of tree
    //Output: Nothing
    Q = new Queue()
    Q.enqueue(root)
    while Q is not empty:
        node = Q.dequeue()
        doSomething(node)
        enqueue node's children
```



`doSomething(node)` puede imprimir, añadir en la lista, decorar vértices, etc.

Recorrido por anchura (BFT): Tree vs. Graph

```
function treeBFT(root):  
    //Input: Root node of tree  
    //Output: Nothing  
    Q = new Queue()  
    Q.enqueue(root)  
    while Q is not empty:  
        node = Q.dequeue()  
        doSomething(node)  
        enqueue node's children
```

```
function graphBFT(start):  
    //Input: start vertex  
    //Output: Nothing  
    Q = new Queue()  
    start.visited = true  
    Q.enqueue(start)  
    while Q is not empty:  
        node = Q.dequeue()  
        doSomething(node)  
        for neighbor in node's adjacent nodes:  
            if not neighbor.visited:  
                neighbor.visited = true  
                Q.enqueue(neighbor)
```

Tenemos que marcar los nodos como visitados, sino bucle infinito.

doSomething(node) puede imprimir, añadir en la lista, decorar vértices, etc.

Recorrido por profundidad

- Para hacer un recorrido DFT en un grafo se puede utilizar una pila en vez de una cola asi como se hace en el BFT
- El recorrido DFT tambien se puede implementar recursivamente

```
function recursiveDFT(node):
    // Input: start node
    // Output: Nothing

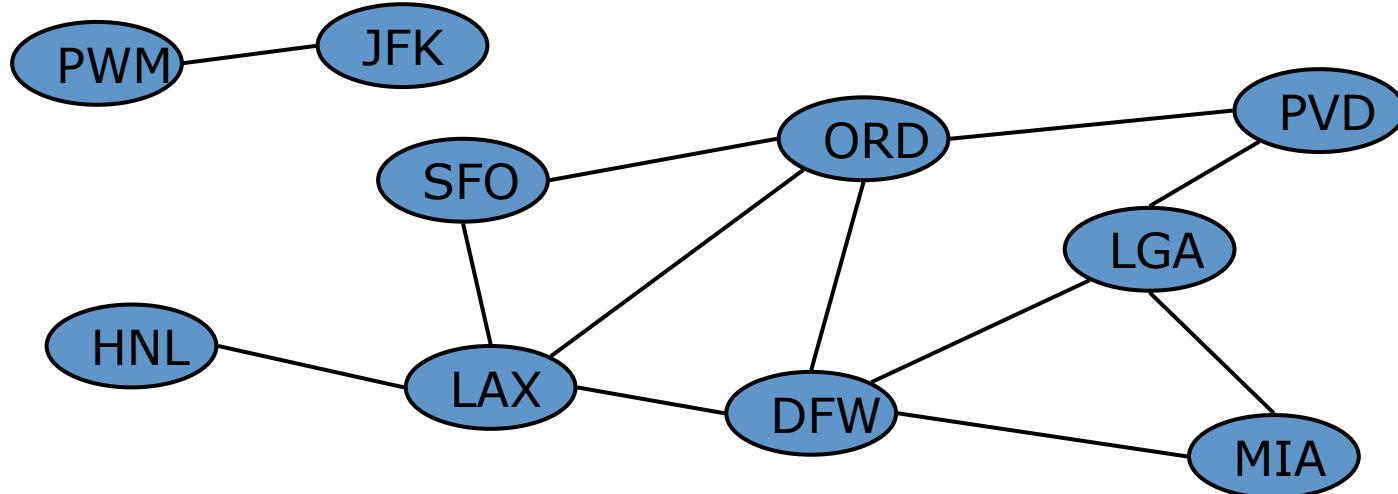
    node.visited = true
    for neighbor in node's adjacent vertices:
        if not neighbor.visited:
            recursiveDFT(neighbor)
```

Aplicaciones

- Red de vuelos aeroes
- Mapa GPS
- Internet
 - Paginas son vértices
 - Links son aristas
- Grafo del Facebook
- Modelaje de estructura moleculares
 - Los atomos son vértices
 - Los enlaces son aristas
- Muchos mas!!

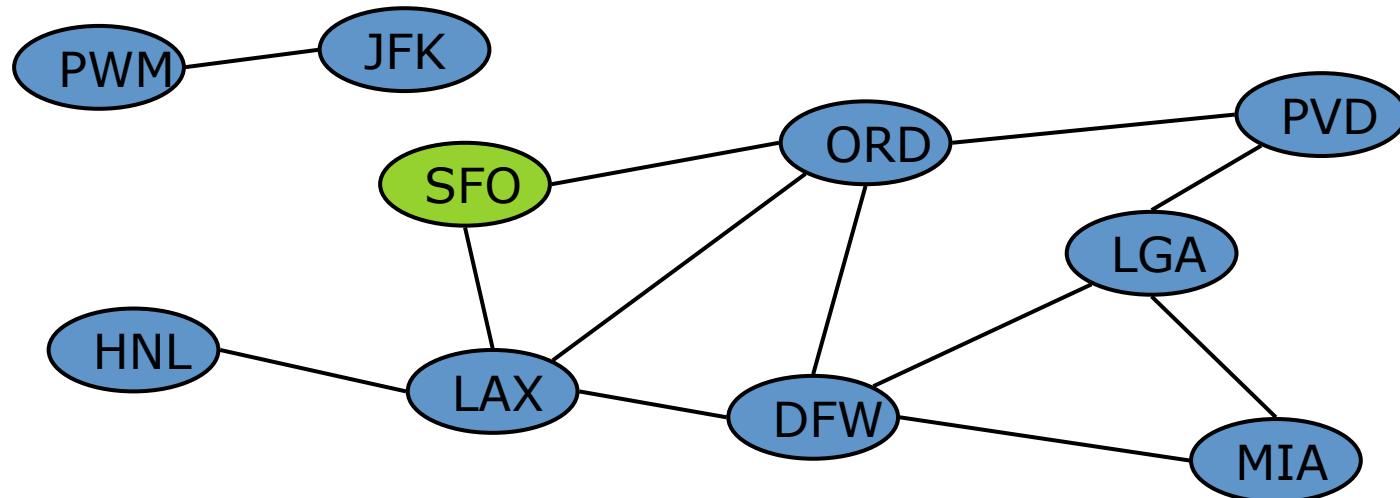
Aplicación: Comprobar si existe una trayectoria aérea

- Problema: Dado un grafo no dirigido con aeropuertos (vértices) y vuelos (aristas), determinar si se puede volar de un lugar a otro.
- Estrategia: Utiliza un recorrido por anchura empezando con el primer nodo, y determina si el nodo de destino ha sido visitado



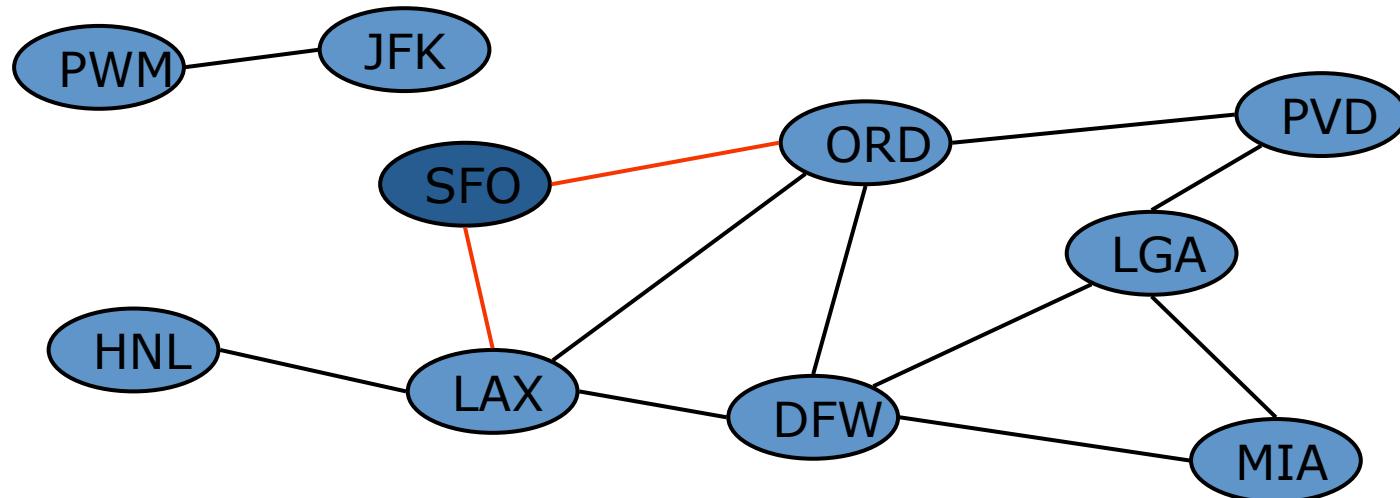
Aplicación: Comprobar si existe una trayectoria aérea

- Problema: Hay algún camino de SFO a PVD?



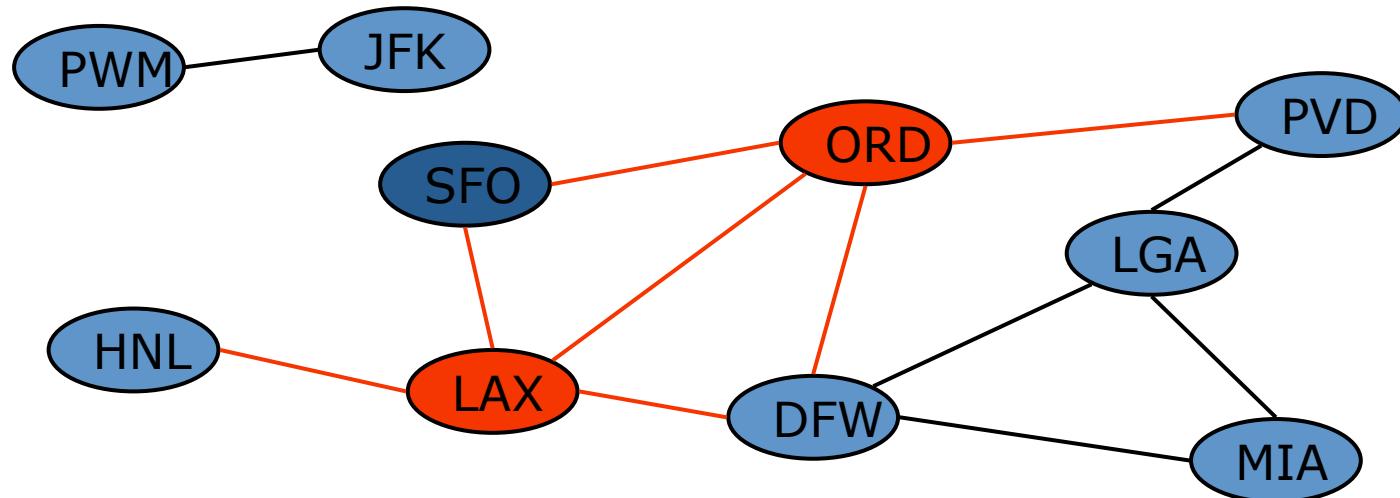
Aplicación: Comprobar si existe una trayectoria aérea

- Problema: Hay algún camino de SFO a PVD?



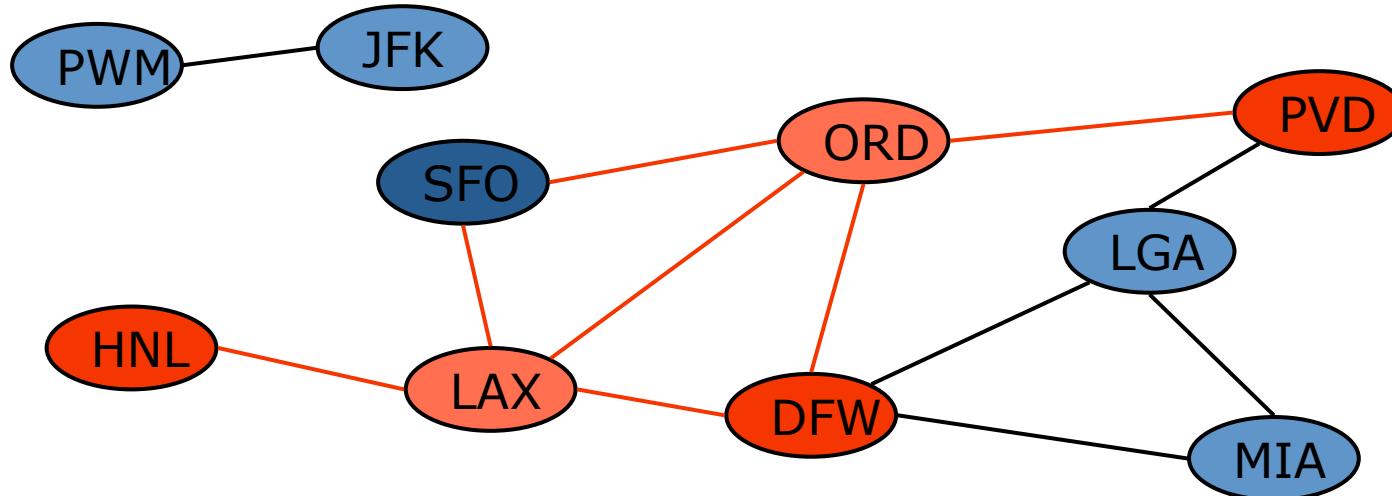
Aplicación: Comprobar si existe una trayectoria aérea

- Problema: Hay algún camino de SFO a PVD?

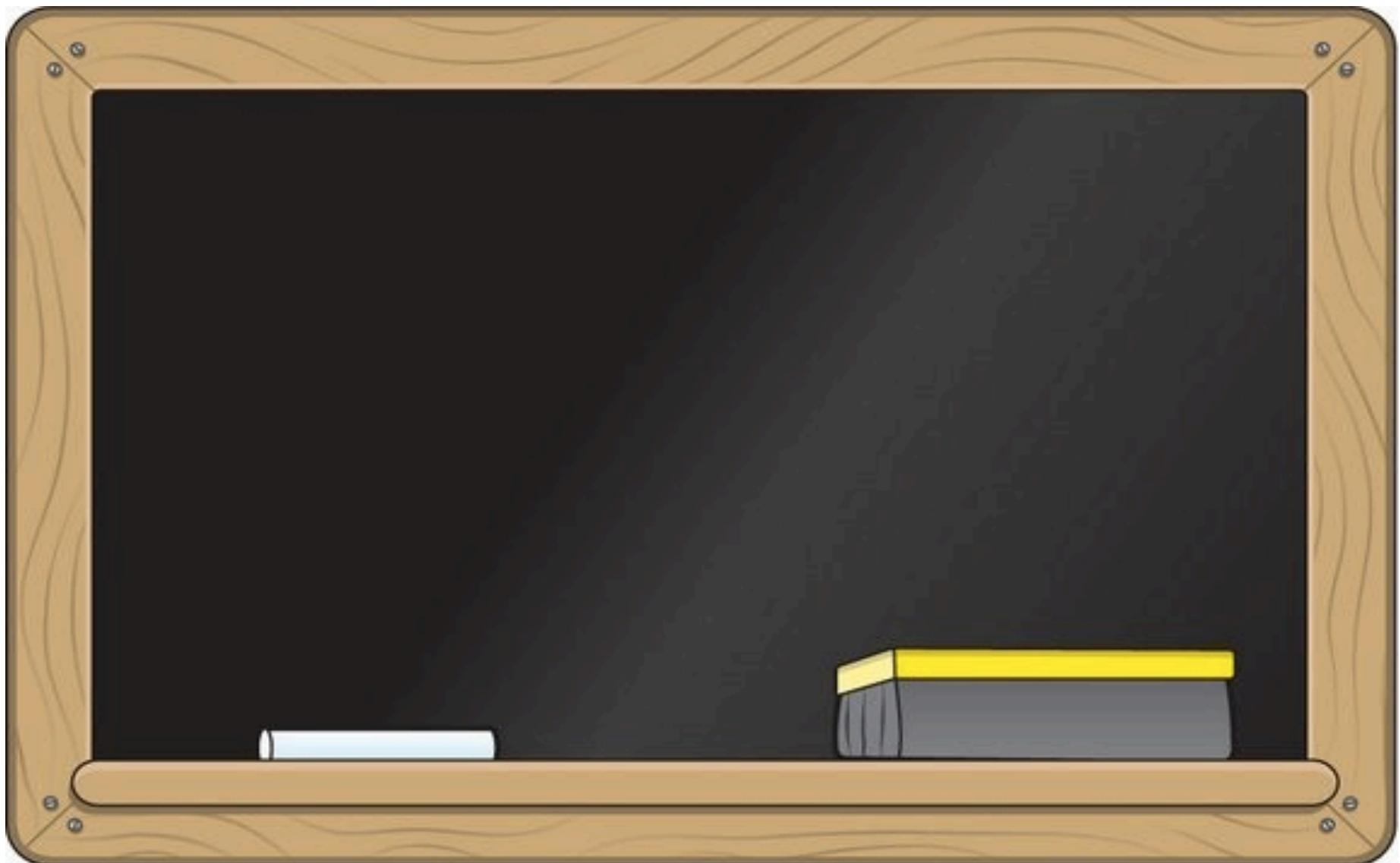


Aplicación: Comprobar si existe una trayectoria aérea

- Problema: Hay algún camino de SFO a PVD?
- **SI!!**, como lo podemos **codificar**?



¿Existe un trayecto aéreo? PseudoCódigo



¿Existe un trayecto aéreo? PseudoCódigo

```
function pathExists(from, to):
    //Input: from: vertex, to: vertex
    //Output: true if path exists, false otherwise

    Q = new Queue()
    from.visited = true
    Q.enqueue(from)
    while Q is not empty:
        airport = Q.dequeue()
        if airport == to:
            return true
        for neighbor in airport's adjacent nodes:
            if not neighbor.visited:
                neighbor.visited = true
                Q.enqueue(neighbor)
    return false
```