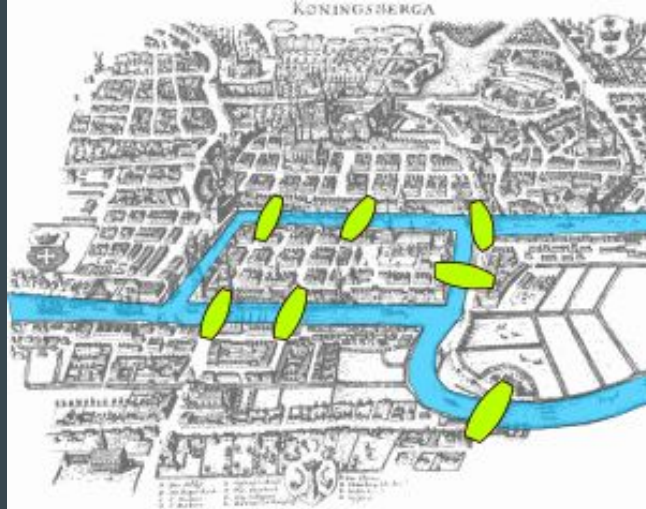


Grafos

...

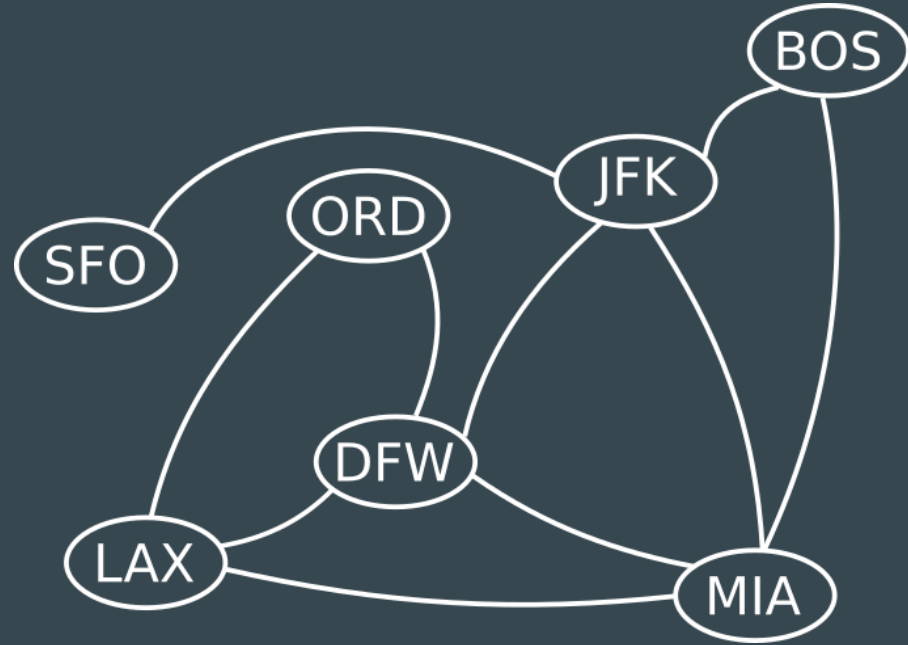
Algoritmos y Estructuras de Datos
2023

La ciudad de Königsberg, en Prusia, tenía 7 puentes. Existe alguna forma de atravesar la ciudad pasando por cada puente una sola vez?



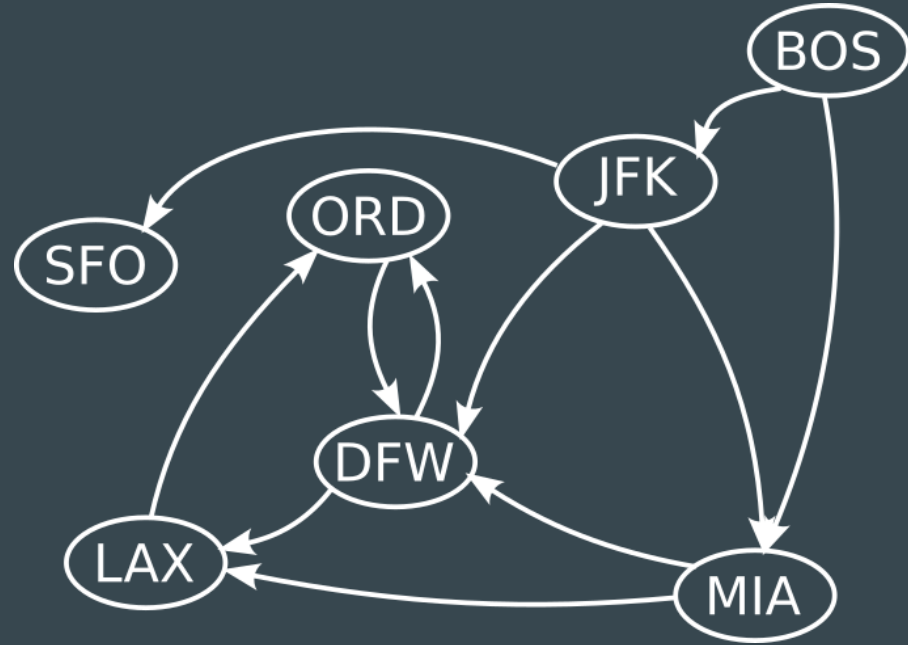
Grafos

- Un grafo es una representación de relaciones entre pares de objetos.
- A cada uno de esos objetos se los denomina **nodo** o **vértice**.
- Las conexiones entre nodos se denominan **aristas** o **arcos**.



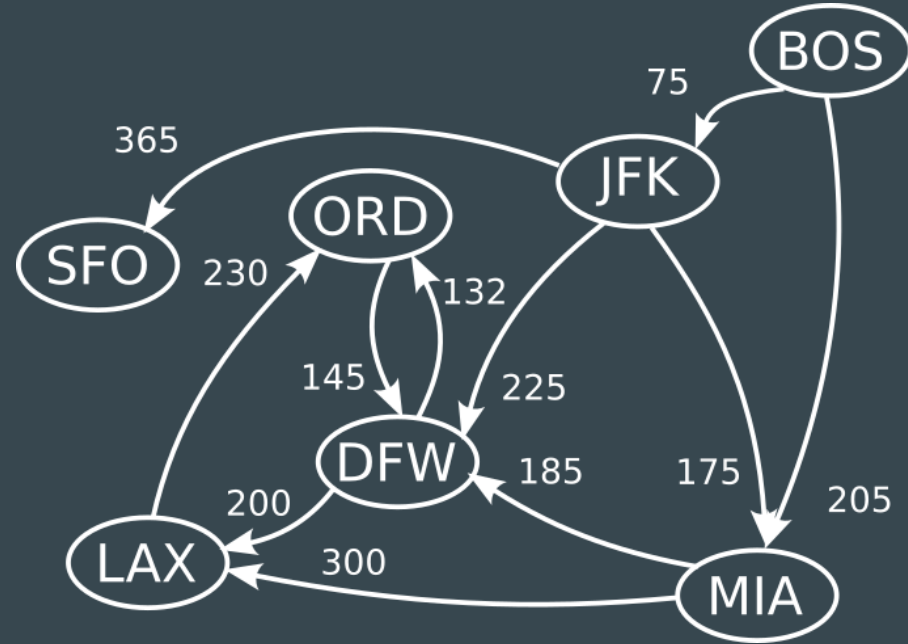
Clasificaciones de grafos

- Las aristas de un grafo pueden ser **dirigidas** si tiene un origen y un destino definido.
- Si no, las aristas se denominan **no dirigidas**.
- El **grafo** será clasificado como **dirigido** o **no dirigido**, dependiendo de las características de sus aristas.



Clasificaciones de grafos

- Además de una dirección, las aristas pueden tener asociados valores llamados **pesos**.
- Dos nodos pueden estar conectados por más de una arista, cada una con su peso determinado.
- Los pesos de que conectan dos nodos en sentidos opuestos también pueden diferir



Caminos

- Un camino es la **secuencia de nodos y aristas** que unen dos nodos particulares.
- En un grafo **dirigido**, si existe un camino que una el nodo u con el nodo v , se dice que “el nodo v es **accesible** desde el nodo u ”.
- Por más que v sea accesible desde u , no necesariamente se da la relación inversa.
- En un grafo no dirigido, se dice que los nodos están **conectados**.

Representaciones

- Existen 3 formas de representación principales:
 - Matrices de adyacencia
 - Listas de adyacencia
 - Listas anidadas
 - Dicionarios
 - Objetos

Matrices de adyacencia

- A cada nodo de un grafo se le asigna un número desde 0 hasta V , siendo V el número de nodos en el grafo.
- Se construye una matriz de tamaño $V * V$, donde:

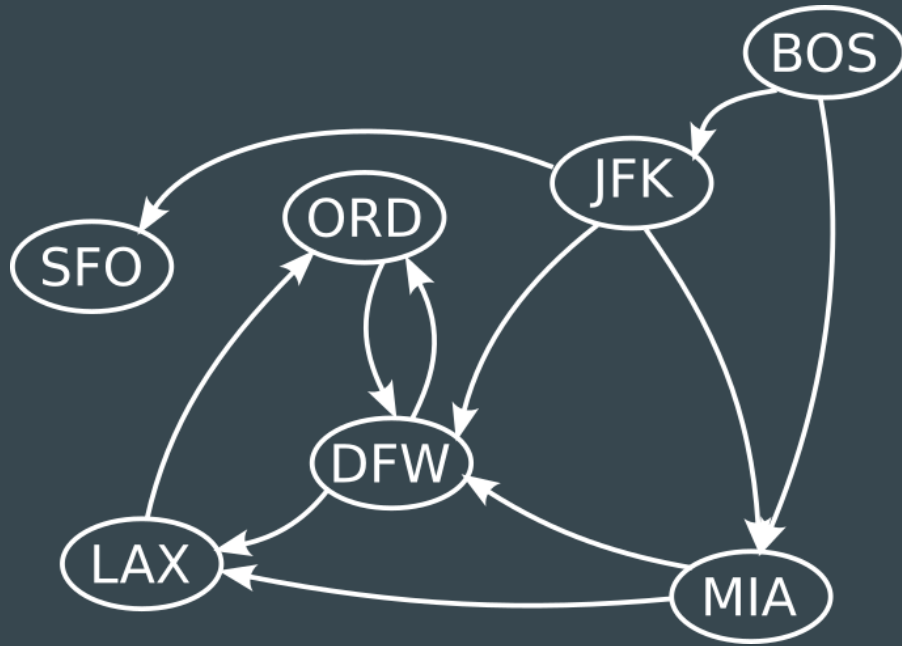
$$A_{ij} \begin{cases} 1 \text{ si } i \text{ y } j \text{ están conectados} \\ 0 \text{ en otro caso} \end{cases}$$

Grafos no pesados

$$A_{ij} \begin{cases} \text{el peso de la unión entre } i \text{ y } j \\ 0 \text{ en otro caso} \end{cases}$$

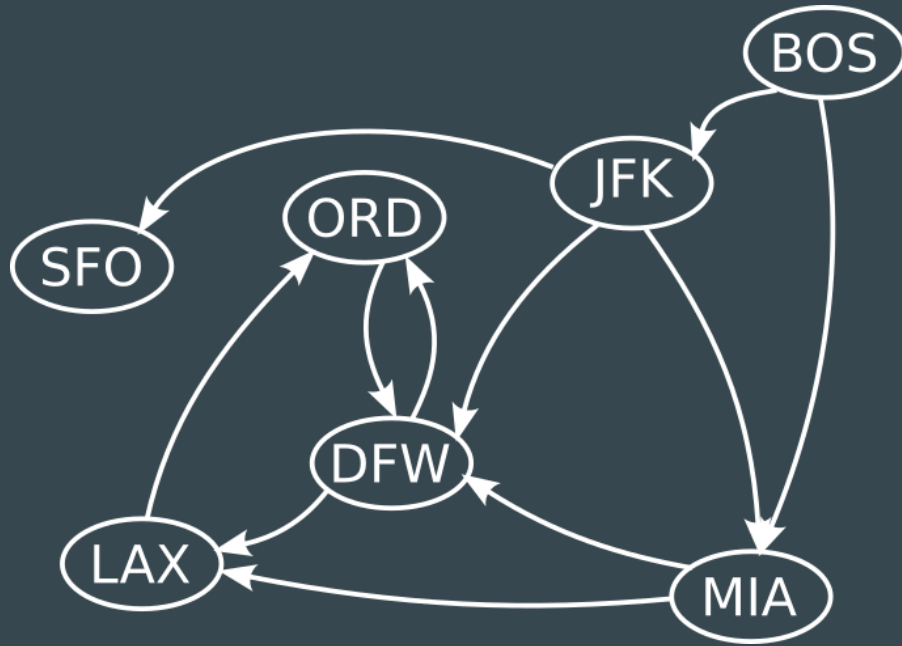
Grafos pesados

Matrices de adyacencia



Indice	Aeropuerto
0	SFO
1	BOS
2	JFK
3	ORD
4	DFW
5	LAX
6	MIA

Matrices de adyacencia



	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1
2	1	0	0	0	1	0	1
3	0	0	0	0	1	0	0
4	0	0	0	1	0	1	0
5	0	0	0	1	0	0	0
6	0	0	0	0	1	1	0

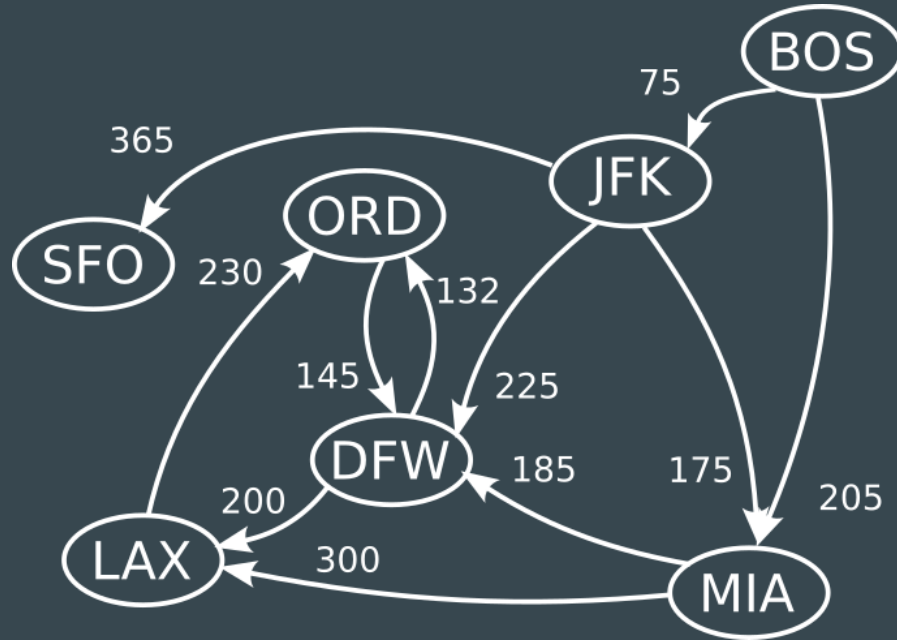
Matrices de adyacencia

```
sfo = [0, 0, 0, 0, 0, 0, 0]
bos = [1, 0, 0, 0, 0, 0, 1]
jfk = [1, 0, 0, 0, 1, 0, 1]
ord = [0, 0, 0, 0, 1, 0, 0]
dfw = [0, 0, 0, 1, 0, 1, 0]
lax = [0, 0, 0, 1, 0, 0, 0]
mia = [0, 0, 0, 0, 1, 1, 0]

matriz_ady = [sfo, bos, jfk, ord,
              dfw, lax, mia]

print("ORD - DFW: ", matriz_ady[3,4])
```

Matrices de adyacencia



```
sfo = [0, 0, 0, 0, 0, 0, 0]
bos = [75, 0, 0, 0, 0, 0, 205]
jfk = [365, 0, 0, 0, 225, 0, 175]
ord = [0, 0, 0, 0, 145, 0, 0]
dfw = [0, 0, 0, 132, 0, 200, 0]
lax = [0, 0, 0, 230, 0, 0, 0]
mia = [0, 0, 0, 0, 185, 300, 0]
```

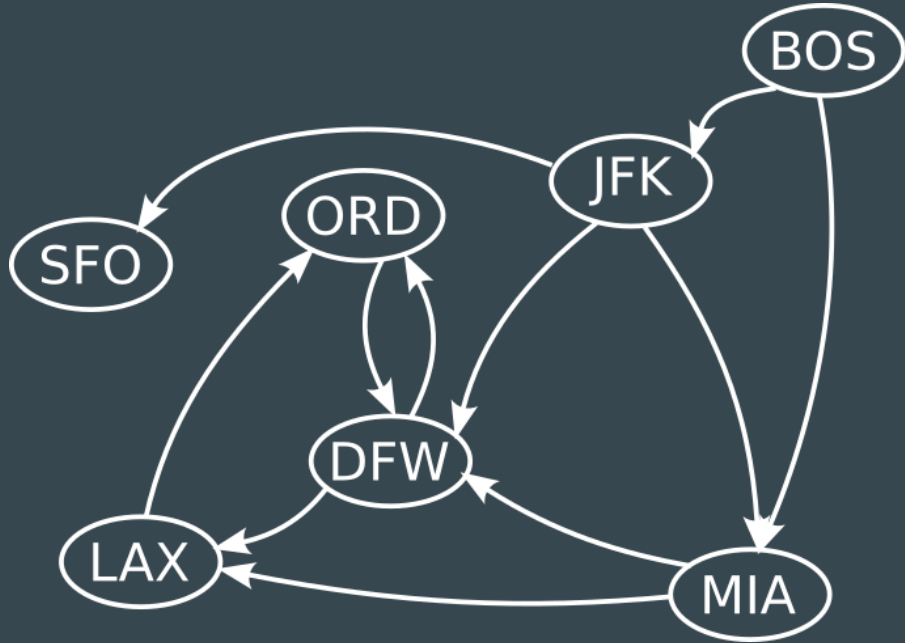
```
matriz_ady = [sfo, bos, jfk, ord,
              dfw, lax, mia]
```

```
print("ORD - DFW: ", matriz_ady[3,4])
```

Listas de adyacencia

- Los grafos también pueden representarse como múltiples listas, una por cada nodo.
- Cada lista posee todos los nodos conectados al nodo correspondiente.
- En Python, dado que no existe nativamente el tipo de dato “matriz”, las implementaciones de matrices y listas de adyacencias son muy similares.

Listas de adyacencia



```
sfo = []  
bos = ["JFK", "MIA"]  
jfk = ["SFO", "DFW", "MIA"]  
ord = ["DFW"]  
dfw = ["ORD", "LAX"]  
lax = ["ORD"]  
mia = ["DFW", "LAX"]
```

```
lista_ady = [sfo, bos, jfk, ord,  
             dfw, lax, mia]
```

```
for i in lista_ady[3]:  
    if i == "DFW":  
        print(True)
```

Listas de adyacencia

- Las listas de adyacencia también pueden implementarse usando diccionarios.
- Esta estrategia hace más simple acceder a los distintos nodos.

```
dicc_ady = { "SFO": {},  
             "BOS": {"JFK": 75, "MIA": 205},  
             "JFK": {"SFO": 365, "DFW": 225, "MIA": 175},  
             "ORD": {"DFW": 145},  
             "DFW": {"ORD": 135, "LAX": 200},  
             "LAX": {"ORD": 230},  
             "MIA": {"DFW": 185, "LAX": 330}  
           }  
  
print("ORD-DFW: ", dicc_ady["ORD"]["DFW"])
```

Objetos

- Otra forma de implementar grafos es definiendo **clases** específicas para nodos y aristas.
- La clase **nodo** contiene el elemento del nodo.
- Dependiendo del tipo de grafo, la clase **arista** va a contener distintos elementos:
 - Origen
 - Destino
 - Peso
 - ...

Recorrido de grafos

- Similar a los árboles, los recorridos pueden ser **por anchura** o **por profundidad**.
- En el recorrido por anchura, se define el nodo de comienzo y se añaden todos los nodos unidos a él a una cola.
- Los siguientes nodos se recorren siguiendo el orden en la cola.
- Se debe llevar la cuenta de qué nodos ya fueron visitados, para no entrar en bucles infinitos.

Recorrido por anchura: pseudocódigo

entrada: grafo, nodo **inicio**.

agregar **inicio** a la **cola** de nodo por visitar

mientras que haya **nodos** en la **cola**:

- quitar **nodo** de la **cola**

- agregar nodo a la lista de **visitados**

- por cada **nodo** conectado a **nodo visitando**:

 - si no fue **visitado**, agregar a la **cola** de nodos por visitar

Recorrido por anchura: código

```
def anchura(grafo, inicio):  
    cola = [inicio]  
    visitados = []  
  
    while len(cola) != 0:  
        visitando = cola.pop(0)  
  
        if visitando not in visitados:  
            visitados.append(visitando)  
  
            for n in grafo[visitando]:  
                cola.append(n)  
  
    return visitados
```

Recorrido por profundidad

- En el recorrido por profundidad, se define el nodo de comienzo y, recursivamente, se visitan todos los nodos conectados a ese inicio.
- Nuevamente, se debe llevar la cuenta de qué nodos ya fueron visitados, para no entrar en bucles infinitos.

Recorrido por profundidad: pseudocódigo

entrada: grafo, nodo **inicio**, lista **visitados**.

agregar **inicio** a **visitados**

por cada **nodo** conectado a **nodo visitando**:

 si **nodo** no está en **visitados**:

 visitar(grafo, nodo, visitados)

devolver **visitados**

Recorrido por profundidad: código

```
def profundidad(grafo, inicio, visitados = []):  
    visitados.append(inicio)  
    for n in grafo[inicio]:  
        if n not in visitados:  
            visitados = profundidad(grafo, n, visitados)  
    return visitados
```

Bibliografía

- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data structures and algorithms in Python. Capítulo 13
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press. Capítulo 22