

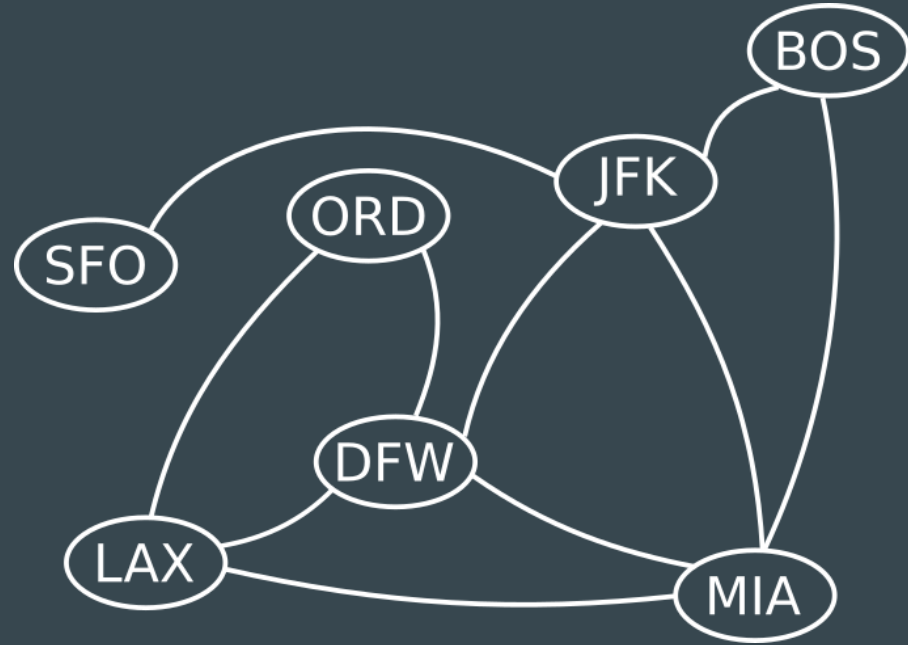
Camino más Corto

...

Algoritmos y Estructuras de Datos
2023

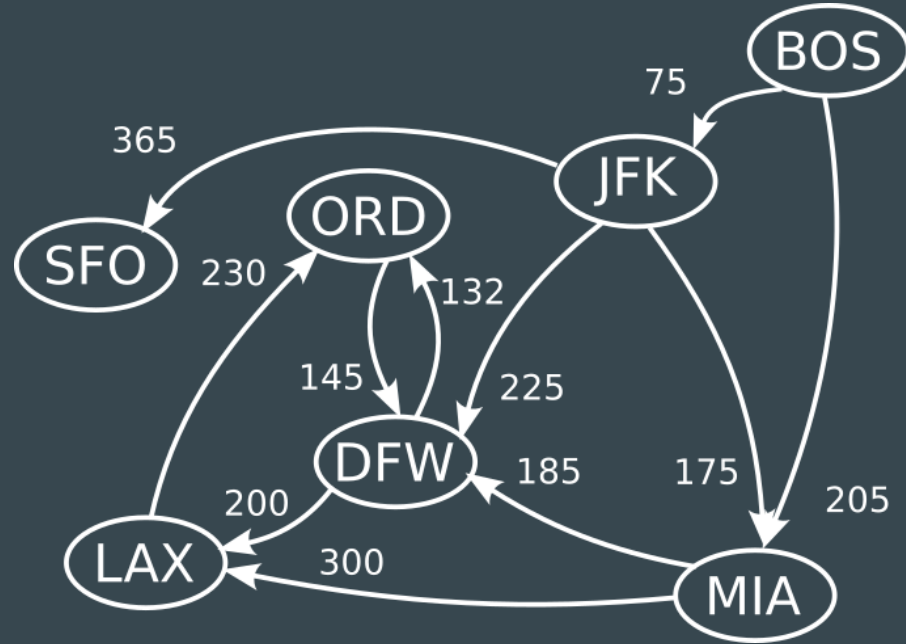
Grafos

- 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

- Si existe una dirección definida en las conexiones de los nodos, el grafo se considera **dirigido**.
- Si las conexiones además tienen algún peso asociado, el grafo se denomina **ponderado** o **pesado**.
- Los pesos de que conectan dos nodos en sentidos opuestos pueden diferir



Representaciones

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

Representaciones

```
sfo = [0, 0, 0, 0, 0, 0, 0]
bos = [0, 0, 75, 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]
```

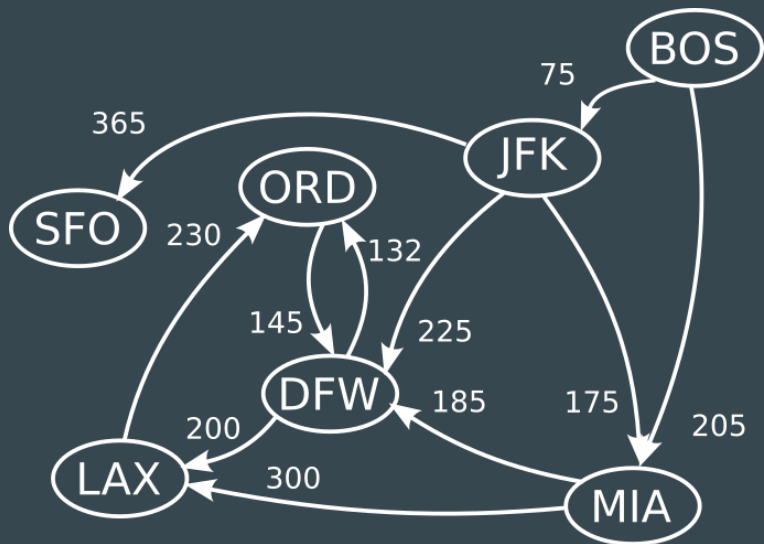
Matrices de adyacencia

```
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}
           }
```

Listas de adyacencia

Caminos

- Un camino es la **secuencia de nodos y aristas** que unen dos nodos particulares.
- El peso de un camino es la suma de todos los pesos que lo componen.



Caminos entre “JFK” y “LAX”:

- 'JFK', 'DFW', 'LAX': 425
- 'JFK', 'MIA', 'DFW', 'LAX': 560
- 'JFK', 'MIA', 'LAX': 505

Camino más corto desde un nodo

- Para calcular todas las distancias de un nodo a los otros, se podría seguir una estrategia de fuerza bruta:
 - Elegir un nodo inicio.
 - Para cada nodo n dentro del grafo, calcular todas las distancias desde inicio hasta n .
 - Elegir la menor distancia dentro de todos los resultados.

Camino más corto desde un nodo

- Una forma de optimizar el código sería aplicando la estrategia de la poda
 - Elegir un nodo inicio.
 - Para cada nodo n dentro del grafo, calcular cada distancia desde inicio hasta n .
 - Si la distancia intermedia es mayor a la distancia menor conocida, no continuar.

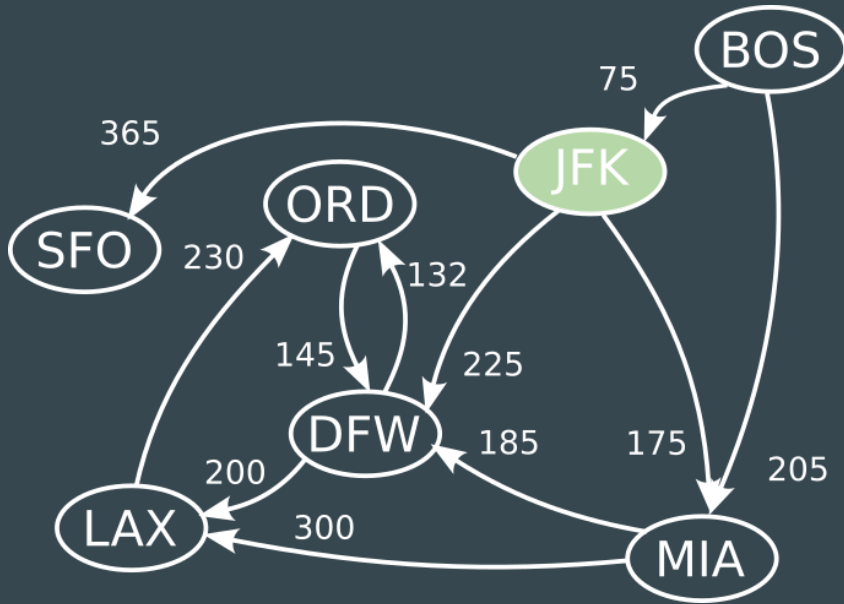
Camino más corto desde un nodo: Dijkstra

- El algoritmo de Dijkstra permite calcular el menor camino desde un inicio a todos los nodos de un grafo.
- Es un **algoritmo ávido** que consiste en calcular las distancias desde un nodo a los adyacentes, y **continuar el cálculo desde el nodo de menor distancia**.
- El algoritmo comienza con el nodo de inicio y termina cuando todos los nodos accesibles fueron visitados.
- Para cada nodo n se almacena la distancia mínima desde el inicio y el nodo anterior que conduce a n .

Algoritmo de Dijkstra: ejemplo

Actual = "JFK"

Visitados = ["JFK"]

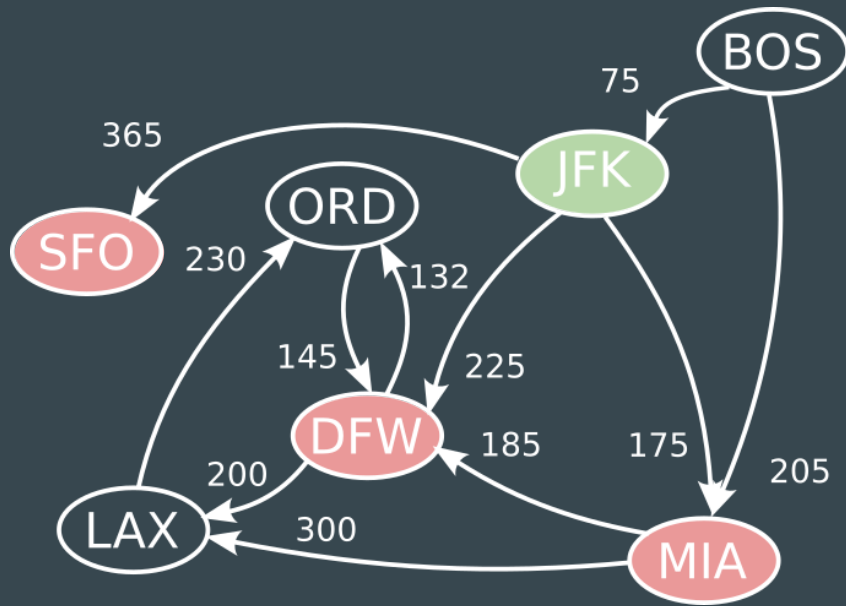


Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	Inf	-
MIA	Inf	-
ORD	Inf	-
DFW	Inf	-
LAX	Inf	-

Algoritmo de Dijkstra: ejemplo

Actual = "JFK"

Visitados = ["JFK"]



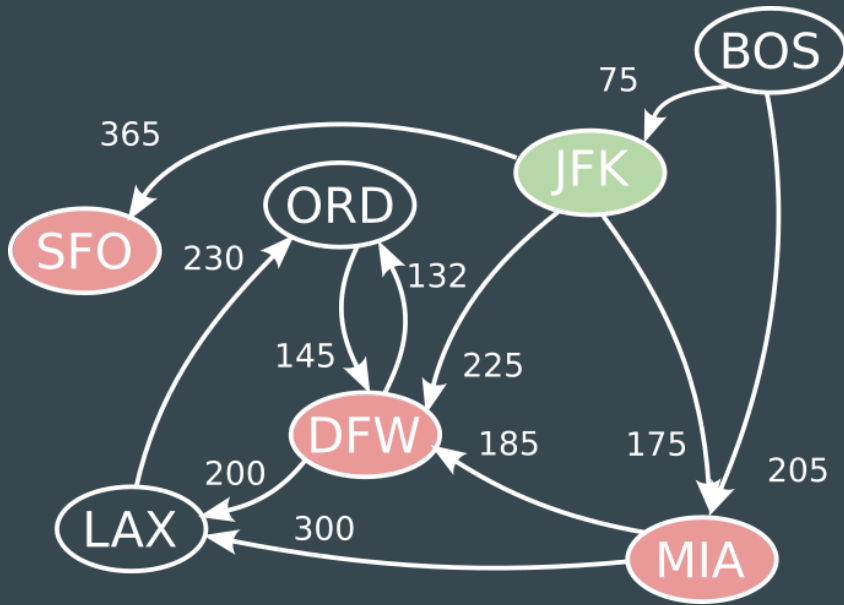
¿ $D_{mia} > D_{jfk-mia}$?
¿ $Inf > 175$?

Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	Inf	-
MIA	Inf	-
ORD	Inf	-
DFW	Inf	-
LAX	Inf	-

Algoritmo de Dijkstra: ejemplo

Actual = "JFK"

Visitados = ["JFK"]

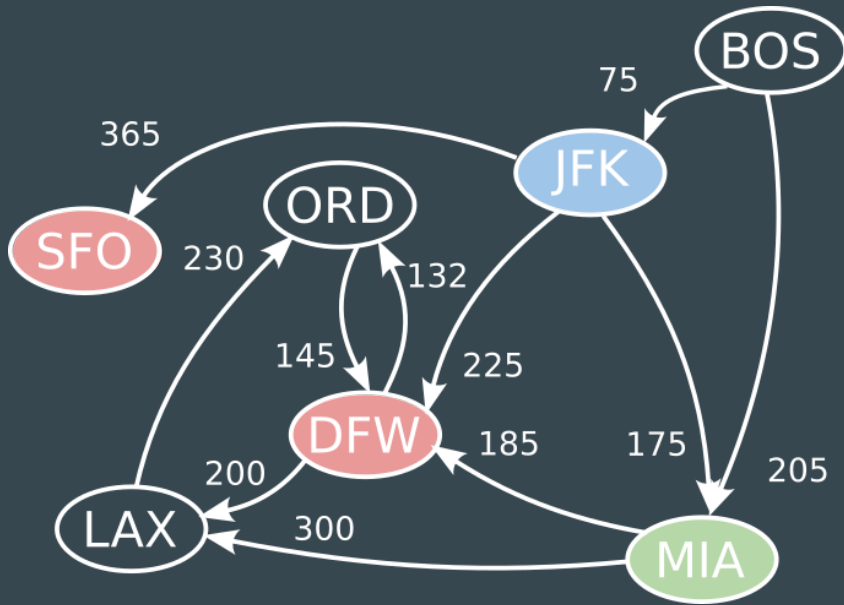


Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	Inf	-
DFW	225	JFK
LAX	Inf	-

Algoritmo de Dijkstra: ejemplo

Actual = "MIA"

Visitados = ["JFK", "MIA"]

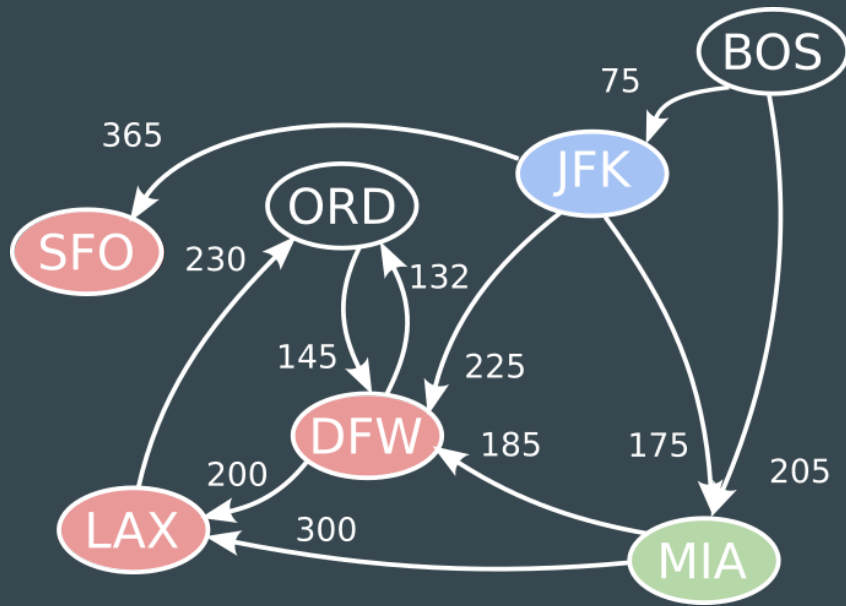


Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	Inf	-
DFW	225	JFK
LAX	Inf	-

Algoritmo de Dijkstra: ejemplo

Actual = "MIA"

Visitados = ["JFK", "MIA"]



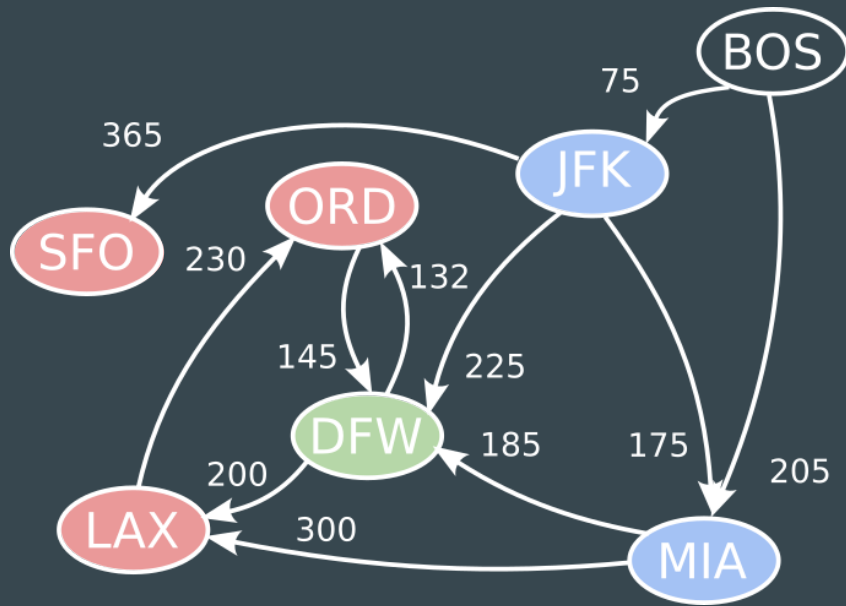
$$\begin{aligned} & \text{¿ } D_{dfw} > D_{jfk-mia} + D_{mia-dfw} ? \\ & \text{¿ } 225 > 175 + 185 ? \end{aligned}$$

Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	Inf	-
DFW	225	JFK
LAX	475	MIA

Algoritmo de Dijkstra: ejemplo

Actual = “DFW”

Visitados = [“JFK”, “MIA”]



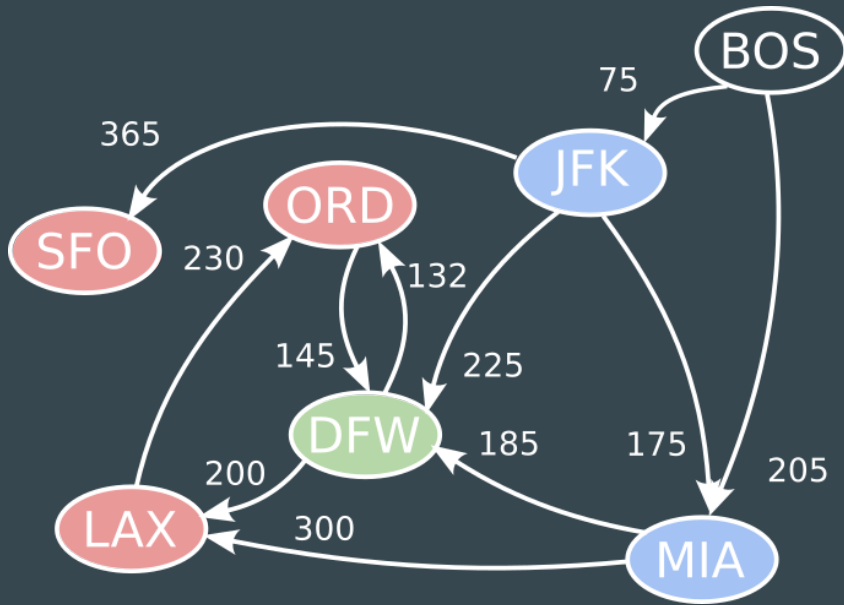
$$\begin{aligned} & \text{¿ } D_{\text{lax}} > D_{\text{jfk-dfw}} + D_{\text{dfw-lax}} ? \\ & \text{¿ } 475 > 225 + 200 ? \end{aligned}$$

Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	132	DFW
DFW	225	JFK
LAX	475	MIA

Algoritmo de Dijkstra: ejemplo

Actual = “DFW”

Visitados = [“JFK”, “MIA”]

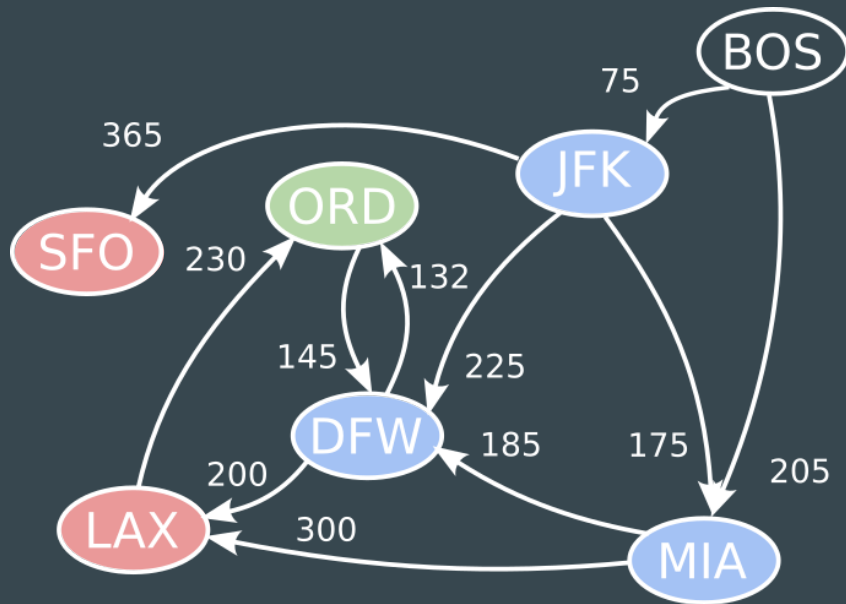


Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	357	DFW
DFW	225	JFK
LAX	445	DFW

Algoritmo de Dijkstra: ejemplo

Actual = "ORD"

Visitados = ["JFK", "MIA", "DFW"]

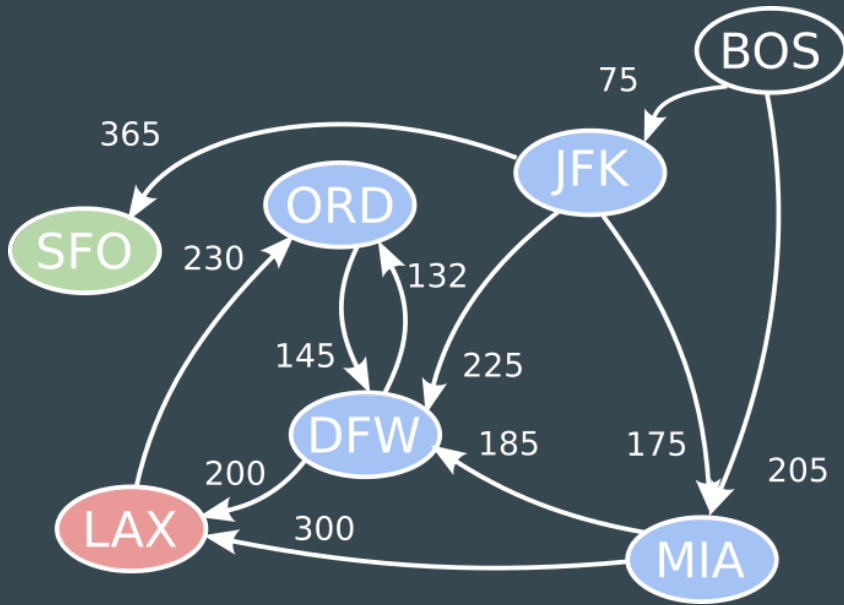


Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	357	DFW
DFW	225	JFK
LAX	445	DFW

Algoritmo de Dijkstra: ejemplo

Actual = "SFO"

Visitados = ["JFK", "MIA", "DFW",
"ORD"]

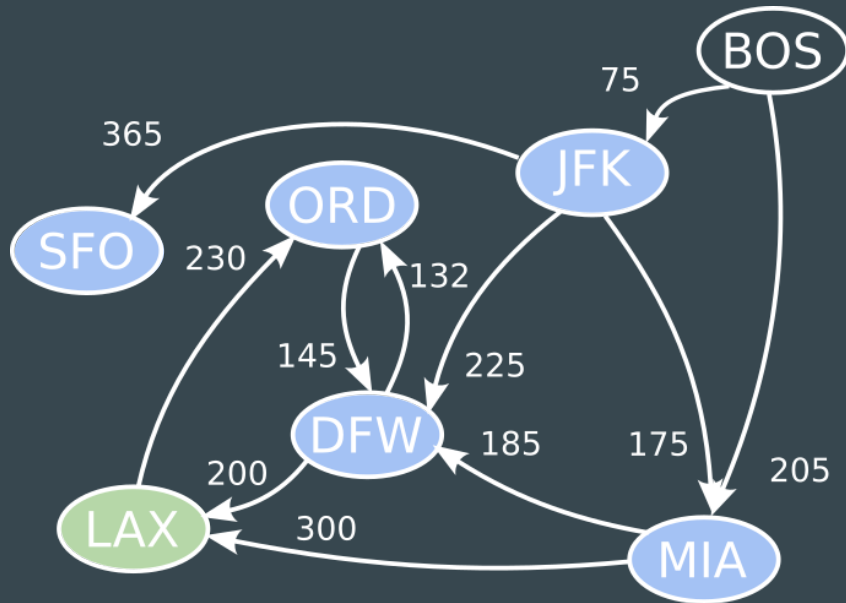


Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	357	DFW
DFW	225	JFK
LAX	445	DFW

Algoritmo de Dijkstra: ejemplo

Actual = "LAX"

Visitados = ["JFK", "MIA", "DFW", "ORD", "SFO"]



Nodo	Distancia	Anterior
JFK	0	None
BOS	Inf	-
SFO	365	JFK
MIA	175	JFK
ORD	357	DFW
DFW	225	JFK
LAX	445	DFW

Algoritmo de Dijkstra: pseudocódigo

- Entrada: **grafo**, nodo **inicio**
- Inicializar tabla de **resultados** con **distancias** y **anteriores**:
 - Si $n == inicio$: distancia = 0 y anterior = None
 - Si no: distancia = Inf y anterior = None
- Inicializar nodo **actual** = inicio
- Inicializar lista **visitados** como lista vacía

Algoritmo de Dijkstra: pseudocódigo

- Mientras que **actual** no sea **None**:
 - Agregar **actual** a **visitados**.
 - Por cada nodo **n** adyacente a **actual**:
 - Calcular distancia desde **n** a **inicio** pasando por **actual** (almacenado en tabla de **resultados**)
 - Si la distancia nueva es menor a la conocida, actualizar valores
 - Elegir siguiente nodo
 - Si existe: menor **distancia** dentro de **resultados** y que no haya sido **visitado**
 - Sino, retornar **None**

Algoritmo de Dijkstra: código

```
def dijkstra(grafo, inicio):
    actual = inicio
    visitados = []
    resultados = {}
    for g in grafo:
        if g == inicio:
            resultados[g] = {"Distancia": 0, "Anterior": None}
        else:
            resultados[g] = {"Distancia": float("inf"), "Anterior": None}

    while actual != None:
        visitados.append(actual)
        for g in grafo[actual]:
            if resultados[g]["Distancia"] > resultados[actual]["Distancia"] + grafo[actual][g]:
                resultados[g]["Distancia"] = resultados[actual]["Distancia"] + grafo[actual][g]
                resultados[g]["Anterior"] = actual
        actual = distancia_menor(resultados, visitados)

    return resultados
```

Algoritmo de Dijkstra: código

```
def distancia_menor(resultados, visitados):  
    dist = float('inf')  
    nodo = None  
    for r in resultados:  
        if resultados[r]["Distancia"] < dist and  
            r not in visitados:  
            dist = resultados[r]["Distancia"]  
            nodo = r  
    return nodo
```


Reconstruyendo el camino mínimo

```
def reconstruir_camino(resultados, inicio, fin):  
    camino = []  
    actual = fin  
    while actual != inicio:  
        camino = [actual] + camino  
        actual = resultados[actual]["Anterior"]  
    camino = [inicio] + camino  
    return camino
```

Distancia mínima entre todos los pares de nodos

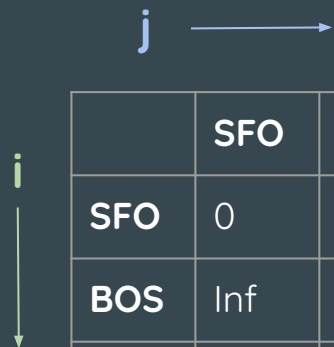
- El algoritmo de Dijkstra permite encontrar las distancias de un nodo particular al resto de los nodos accesibles.
- Si se quisieran conocer todas las distancias, se debería ejecutar el código n veces.
- Una alternativa para disminuir el número de cálculos es el algoritmo de **Floyd**.

Algoritmo de Floyd

- El algoritmo de Floyd utiliza una **matriz de adyacencia** para calcular las distancias entre todos los pares de nodos.
- Este algoritmo tiene muchas coincidencias con el algoritmo de **Dijkstra**.
- Para facilitar los cálculos, al comienzo las **distancias entre nodos no adyacentes** se establece como **infinita**.
- Luego se recorre la **matriz** completa y se comprueba si la distancia conocida entre dos nodos es mayor a la distancia entre esos nodos, pasando por un nodo adicional:

$$¿ D_{ij} > D_{ik} + D_{kj} ?$$

Algoritmo de Floyd: ejemplo



	SFO	BOS	JFK	ORD	DFW	LAX	MIA
SFO	0	Inf	Inf	Inf	Inf	Inf	Inf
BOS	Inf	0	75	Inf	Inf	Inf	205
JFK	365	Inf	0	Inf	225	Inf	175
ORD	Inf	Inf	Inf	0	145	Inf	Inf
DFW	Inf	Inf	Inf	132	0	200	Inf
LAX	Inf	Inf	Inf	230	Inf	0	Inf
MIA	Inf	Inf	Inf	Inf	185	300	0

Calculando la distancia mínima entre **BOS** y **SFO**:

$i = 1; j = 0; D_{10} = \text{Inf}$

$k = 0: D_{10} = \text{Inf}; D_{00} = 0 \Rightarrow$ No cambiar
 $k = 1: D_{11} = 0; D_{10} = \text{Inf} \Rightarrow$ No cambiar
 $k = 2: D_{12} = 75; D_{20} = 365 \Rightarrow$ **Cambiar**
Dij = 440

$k = 3: D_{13} = \text{Inf}; D_{30} = \text{Inf} \Rightarrow$ No cambiar
 $k = 4: D_{14} = \text{Inf}; D_{40} = \text{Inf} \Rightarrow$ No cambiar
 $k = 5: D_{15} = \text{Inf}; D_{50} = \text{Inf} \Rightarrow$ No cambiar
 $k = 6: D_{16} = 205; D_{60} = \text{Inf} \Rightarrow$ No cambiar

Algoritmo de Floyd: ejemplo

j →

i ↓

	SFO	BOS	JFK	ORD	DFW	LAX	MIA
SFO	0	Inf	Inf	Inf	Inf	Inf	Inf
BOS	Inf	0	75	Inf	Inf	Inf	205
JFK	365	Inf	0	Inf	225	Inf	175
ORD	Inf	Inf	Inf	0	145	Inf	Inf
DFW	Inf	Inf	Inf	132	0	200	Inf
LAX	Inf	Inf	Inf	230	Inf	0	Inf
MIA	Inf	Inf	Inf	Inf	185	300	0

Calculando la distancia mínima entre **JFK** y **LAX**:

$i = 2; j = 5; D_{25} = \text{Inf}$

$k = 0: D_{20} = 365; D_{05} = \text{Inf} \Rightarrow$ No cambiar

$k = 1: D_{21} = \text{Inf}; D_{15} = \text{Inf} \Rightarrow$ No cambiar

$k = 2: D_{22} = 0; D_{25} = \text{Inf} \Rightarrow$ No cambiar

$k = 3: D_{23} = \text{Inf}; D_{35} = \text{Inf} \Rightarrow$ No cambiar

$k = 4: D_{24} = 225; D_{45} = 200 \Rightarrow$ **Cambiar**

Dij = 425

$k = 5: D_{25} = \text{Inf}; D_{55} = \text{Inf} \Rightarrow$ No cambiar

$k = 6: D_{26} = 175; D_{65} = 300 \Rightarrow$ No cambiar

Algoritmo de Floyd: pseudocódigo

- Entrada: **matriz** de adyacencia con **0**, **distancias** e **infinitos**
- Para cada nodo intermedio (k):
 - Para cada nodo de inicio (i):
 - Para cada nodo de fin (j):
 - Calcular D_{ij} y la suma entre D_{ik} y D_{kj}
 - Si corresponde, actualizar valores

Algoritmo de Floyd: código

```
def preparar_matriz(matriz):  
    for i in range (0, len(matriz)):  
        for j in range (0, len(matriz)):  
            if i != j and matriz[i][j] == 0:  
                matriz[i][j] = float("inf")  
    return matriz  
  
def floyd(matriz):  
    for k in range (0, len(matriz)):  
        for i in range (0, len(matriz)):  
            for j in range (0, len(matriz)):  
                if matriz[i][j] > matriz[i][k] + matriz[k][j]:  
                    matriz[i][j] = matriz[i][k] + matriz[k][j]  
    return matriz
```