🟦 Warshall-Algorithmus (Floyd-Warshall)

1️⃣ Grundidee

Der **Warshall-Algorithmus** (häufig als **Floyd-Warshall** bezeichnet) berechnet die **kürzesten Wege zwischen allen Knotenpaaren** eines Graphen.

- Dynamic-Programming-Ansatz
- Betrachtet schrittweise alle Knoten als mögliche Zwischenknoten
- Funktioniert für **gewichtete Graphen**
- **Negative Kantengewichte erlaubt**, aber **keine negativen Zyklen**
- ist **nicht** greedy wie Dijkstra

2️⃣ Voraussetzungen

- Gewichteter Graph
- Darstellung als **Adjazenzmatrix**
- Keine negativen Zyklen

3️⃣ Laufzeiten & Eigenschaften

Eigenschaft	Wert
Laufzeit	$O(V^3)$
Speicherbedarf	$O(V^2)$
Kürzeste Wege	alle Paare
Negative Kanten	✔️ erlaubt
Negative Zyklen	❌ nicht erlaubt

Hinweis: Der Algorithmus ist unabhängig von der Anzahl der Kanten – nur die Anzahl der Knoten zählt.

4️⃣ Idee des Algorithmus

Wir prüfen für jedes Knotenpaar (i, j), ob der Weg über einen Zwischenknoten k kürzer ist:

```
dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Dies wird für alle k von 1 bis V durchgeführt.

5️⃣ Schritt-für-Schritt-Beispiel

Graph (Adjazenzmatrix):

	A	B	C
A	0	3	∞
B	∞	0	1
C	2	∞	0

Zwischenknoten B

- Prüfe A → B → C: 3 + 1 = 4 < ∞ → Update

Ergebnis:

	A	B	C
A	0	3	4
B	∞	0	1
C	2	∞	0

6️⃣ Besonderheiten / Prüfungsrelevante Hinweise

- Warshall berechnet **alle kürzesten Wege**
- Sehr häufige Prüfungsfrage:
  - *Unterschied Dijkstra vs. Warshall?*
- Negative Kantengewichte erlaubt
- Negative Zyklen müssen ausgeschlossen sein

7️⃣ Vor- und Nachteile

Vorteile

- alle kürzesten Wege auf einmal
- einfacher, kompakter Algorithmus
- negative Kanten erlaubt

Nachteile

- hohe Laufzeit $O(V^3)$
- ungeeignet für große Graphen

🧠 Merksatz für die Prüfung

Der Warshall-Algorithmus berechnet alle kürzesten Wege mit dynamischer Programmierung und erlaubt negative Kantengewichte ohne negative Zyklen.

🎱 Python-Implementierung

In [1]:
```python
def floyd_warshall(dist):
    n = len(dist)

    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist


# Beispiel
INF = float("inf")

dist = [
    [0,   3,   INF],
    [INF, 0,   1],
    [2,   INF, 0]
]

result = floyd_warshall(dist)

for row in result:
    print(row)
```

```
[0, 3, 4]
[3, 0, 1]
[2, 5, 0]
```