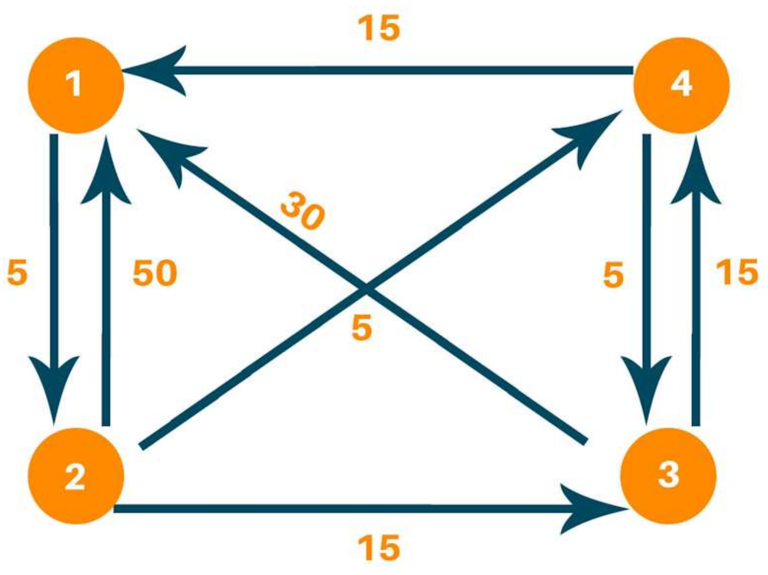
In [1]:
```python
# Styling notebook
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]:

Floyd-Warshall APSP (All Pair Shortest Path) Algorithm



APSP means it computes the lengths of the shortests path between every pair of certices in a labeled graph.

[https://favtutor.com/blogs/floyd-warshall-algorithm (https://favtutor.com/blogs/floyd-warshall-algorithm)](https://favtutor.com/blogs/floyd-warshall-algorithm)

For comparision: SSSP (Single Source Shortest Path) computes the shortest distances given a specific start vertex (source).

In general, SSSP (something like "Dijkstra's Algorithm") is faster (I mean, it just has to compute less distances!) and can be used in practical shortest path problems. Floyd-Warshall is just more fun to figure out - it's stunningly simple!

In [2]:
```python
# Floyd-Warshall (all pair shortest path) in all its glorious simplicity!
# AM is the Adjacency Matrix of the graph
def floydWarshall(AM) :
    numVertices = len(AM)

    # Main FW Loop (Yep, that's all there is, really!)
    for i in range(numVertices) :
        for j in range(numVertices) :
            for k in range(numVertices) :
                AM[j][k] = min(AM[j][k], AM[j][i] + AM[i][k])


# Printing the output (Almost as long as the main algorithm)
def printMatrix(AM) :
    numVertices = len(AM)
    for i in range(numVertices) :
        for j in range(numVertices) :
            print(AM[i][j], end="  ")
        print(" ")
```

In [3]:
```
# Adjacency Matrix (see graph above)
# 9999 stands for "not reachable directly"
# The main diagonal is all 0 for obvious (?) reasons
AM = [[0, 5, 9999, 9999],
      [50, 0, 15, 5],
      [30, 9999, 0, 15],
      [15, 9999, 5, 0]]

# And off you go!
floydWarshall(AM)
printMatrix(AM)
```

```
0  5  15  10
5  0  10  5
30  35  0  15
15  20  5  0
```

Try understanding the result. For example the 35 in the third row, second column, means that the shortest path between vertex 3 and vertext 2 has a length of 35. There are actually two paths with this length. Find them!

Ok, that doesn't give you the real shortest path, just the length. Consult your A&DS instructor on how to also find the paths - it's a little tricky!

Or maybe you can find out by studying the code here (no explanations, that's A&DS stuff)

In [6]:
```
def floydWarshall2(AM) :
    numVertices = len(AM)

    # sep[i][j] is a vertex that is on the shortest path between vertex i and ver
    # initialized with -1, so we see if there is such a separating vertex
    sep = [[-1 for y in range(numVertices)] for x in range(numVertices)]

    # Main FW loop (Yep, that's all there is, really!)
    for i in range(numVertices) :
        for j in range(numVertices) :
            for k in range(numVertices) :
                # We have to rewrite this a bit:
                if AM[j][i] + AM[i][k] < AM[j][k] : # Shorter path over the separ
                    AM[j][k] = AM[j][i] + AM[i][k]  # YEP!
                    sep[j][k] = i                   # Shortest path must lead ove
    return sep

# Reconstruct the path from i to j via the separation matrix
def path (sep, i, j) :
    if sep[i][j] == -1 :
        print (i, "-->", j)
    else :
        sepVertex = sep[i][j]
        path(sep,i,sepVertex)
        path(sep,sepVertex,j)
```

In [13]:
```python
AM = [[0, 5, 9999, 9999],
      [50, 0, 15, 5],
      [30, 9999, 0, 15],
      [15, 9999, 5, 0]]

sep = floydWarshall2(AM)
print("Distance Matrix")
printMatrix(AM)
print("\nSeparator Matrix")
printMatrix(sep)


print("\npath(sep,0,1)")
path(sep,0,1) # Path from index 0 (vertex "1") to index 1 (vertex "2")
print("\npath(sep,1,0)")
path(sep,1,0) # Path from index 1 (vertex "2") to index 0 (vertex "1")
print("\npath(sep,0,2)")
path(sep,0,2) # Path from index 0 (vertex "1") to index 2 (vertex "3")
```

```
Distance Matrix
0  5  15  10
20  0  10  5
30  35  0  15
15  20  5  0

Separator Matrix
-1  -1  3  1
3  -1  3  -1
-1  0  -1  -1
-1  0  -1  -1

path(sep,0,1)
0 --> 1

path(sep,1,0)
1 --> 3
3 --> 0

path(sep,0,2)
0 --> 1
1 --> 3
3 --> 2
```