Plain `algo` and code

title: "Floyd-Warshall" (algo only)
parameters: ("V", "E", "w") (algo only)

```
FLOYD-WARSHALL(V, E, w):
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                    // edge weights
For v in V:
    dist[v, v] ← 0                          // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

```python
def floyd_warshall(G):
  # let G be an adjacency matrix
  dist = G

  for k in range(len(G)):
    for i in range(len(G)):
      for j in range(len(G)):
        if dist[i][j] > dist[i][k] + dist[k][j]:
          dist[i][j] = dist[i][k] + dist[k][j]

  return dist
```

Basic styling parameters

fill:	none
stroke:	2pt + black
radius:	10pt
row-gutter:	8pt
column-gutter:	8pt
inset:	15pt
indent-size:	12pt (algo only)
indent-guides:	1pt + gray
indent-guides-offset:	4pt
comment-prefix:	[#sym.triangle] (algo only)

FLOYD-WARSHALL(V, E, w):

1 **Let** $\text{dist}[u, v] \leftarrow \infty$ **for** u, v **in** V
2 **For** (u, v) **in** E**:**
3 $\quad \text{dist}[u, v] \leftarrow w(u, v)$ ▷ edge weights
4 **For** v **in** V**:**
5 $\quad \text{dist}[v, v] \leftarrow 0$ ▷ base case
6
7 **For** $k \leftarrow 1$ **to** $|V|$**:**
8 $\quad$ **For** $i \leftarrow 1$ **to** $|V|$**:**
9 $\quad\quad$ **For** $j \leftarrow 1$ **to** $|V|$**:**
10 $\quad\quad\quad$ ▷ if new path is shorter, reduce distance
11 $\quad\quad\quad$ **If** $\text{dist}[i, j] > \text{dist}[i, k] + \text{dist}[k, j]$**:**
12 $\quad\quad\quad\quad \text{dist}[i, j] \leftarrow \text{dist}[i, k] + \text{dist}[k, j]$
13
14 **Return** dist

```python
def floyd_warshall(G):
    # let G be an adjacency matrix
    dist = G

    for k in range(len(G)):
        for i in range(len(G)):
            for j in range(len(G)):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
```

Empty bodies

`code` with empty raw text

`code` with empty raw block

code with non-sequence raw block

```
def floyd_warshall(G):
  # let G be an adjacency matrix
  dist = G

  for k in range(len(G)):
    for i in range(len(G)):
      for j in range(len(G)):
        if dist[i][j] > dist[i][k] + dist[k][j]:
          dist[i][j] = dist[i][k] + dist[k][j]

  return dist
```

Indent guides with line wrapping

indent-guides: 1pt + black

```
FLOYD-WARSHALL(V, E, w):
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                                    // edge weights
For v in V:
    dist[v, v] ← 0                                          // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]
                blah blah blah blah blah blah blah blah blah blah blah
                blah blah blah blah blah

Return dist
```

```
def floyd_warshall(G):
    # let G be an adjacency matrix
    dist = G

    for k in range(len(G)):
        for i in range(len(G)):
            for j in range(len(G)):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    blah blah blah blah blah blah blah blah blah blah blah blah blah blah
blah blah blah

    return dist
```

code indent guides with custom tab size

indent-guides: 1pt + black

tab-size: 2

```
def floyd_warshall(
    G
):
  # let G be an adjacency matrix
  dist = G

  for k in range(len(G)):
    for i in range(len(G)):
      for j in range(len(G)):
        if dist[i][j] > dist[i][k] + dist[k][j]:
          dist[i][j] = dist[i][k] + dist[k][j]

  return dist
```

No line numbers

line-numbers: false

```
FLOYD-WARSHALL(V, E, w):
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                    // edge weights
For v in V:
    dist[v, v] ← 0                          // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

```python
def floyd_warshall(G):
  # let G be an adjacency matrix
  dist = G

  for k in range(len(G)):
    for i in range(len(G)):
      for j in range(len(G)):
        if dist[i][j] > dist[i][k] + dist[k][j]:
          dist[i][j] = dist[i][k] + dist[k][j]

  return dist
```

`algo` without keywords

keyword-styles: none

```
FLOYD-WARSHALL(V, E, w):
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                          // edge weights
For v in V:
    dist[v, v] ← 0                                // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

`algo` with custom keywords

keywords: ("in", "to")

```
FLOYD-WARSHALL(V, E, w):
    Let dist[u, v] ← ∞ for u, v in V
    For (u, v) in E:
        dist[u, v] ← w(u, v)                    // edge weights
    For v in V:
        dist[v, v] ← 0                          // base case

    For k ← 1 to |V|:
        For i ← 1 to |V|:
            For j ← 1 to |V|:
                // if new path is shorter, reduce distance
                If dist[i, j] > dist[i, k] + dist[k, j]:
                    dist[i, j] ← dist[i, k] + dist[k, j]

    Return dist
```

`algo` without title

title: none

```
(V, E, w):
    Let dist[u, v] ← ∞ for u, v in V
    For (u, v) in E:
        dist[u, v] ← w(u, v)                          // edge weights
    For v in V:
        dist[v, v] ← 0                                // base case

    For k ← 1 to |V|:
        For i ← 1 to |V|:
            For j ← 1 to |V|:
                // if new path is shorter, reduce distance
                If dist[i, j] > dist[i, k] + dist[k, j]:
                    dist[i, j] ← dist[i, k] + dist[k, j]

    Return dist
```

`algo` without parameters

parameters: ()

```
FLOYD-WARSHALL():
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                          // edge weights
For v in V:
    dist[v, v] ← 0                                // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

`algo` without header

title: none

parameters: ()

```
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                          // edge weights
For v in V:
    dist[v, v] ← 0                                // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

`algo` with content-type parameters

parameters: ([#text(blue, [V])], [#text(red, [E])], [#text(green, [w])])

FLOYD-WARSHALL(V, E, w):

```
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                    // edge weights
For v in V:
    dist[v, v] ← 0                          // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

`algo` with content-type title

title: [#set text(red);Floyd-Warshall]

Floyd-Warshall():

```
1  Let dist[u, v] ← ∞ for u, v in V
2  For (u, v) in E:
3      dist[u, v] ← w(u, v)                          // edge weights
4  For v in V:
5      dist[v, v] ← 0                                // base case
6
7  For k ← 1 to |V|:
8      For i ← 1 to |V|:
9          For j ← 1 to |V|:
10             // if new path is shorter, reduce distance
11             If dist[i, j] > dist[i, k] + dist[k, j]:
12                 dist[i, j] ← dist[i, k] + dist[k, j]
13
14 Return dist
```

`algo` with custom header

Floyd-Warshall Algorithm

Inputs: graph $G = (V, E)$
weight function $w : E \to \mathbb{R}$

Outputs: distance matrix dist

Let $\text{dist}[u, v] \leftarrow \infty$ **for** u, v **in** V
For (u, v) **in** E**:**
    $\text{dist}[u, v] \leftarrow w(u, v)$ // edge weights
For v **in** V**:**
    $\text{dist}[v, v] \leftarrow 0$ // base case

For $k \leftarrow 1$ **to** $|V|$**:**
    For $i \leftarrow 1$ **to** $|V|$**:**
        For $j \leftarrow 1$ **to** $|V|$**:**
            // if new path is shorter, reduce distance
            If $\text{dist}[i, j] > \text{dist}[i, k] + \text{dist}[k, j]$**:**
                $\text{dist}[i, j] \leftarrow \text{dist}[i, k] + \text{dist}[k, j]$

Return dist

Text styling

main-text-styles: x => text(fill: green)[#x]

line-number-styles: x => text(fill: red)[#x]

comment-styles: x => text(fill: blue, x) (algo only)

FLOYD-WARSHALL(V, E, w):

```
1  Let dist[u, v] ← ∞ for u, v in V
2  For (u, v) in E:
3      dist[u, v] ← w(u, v)                          // edge weights
4  For v in V:
5      dist[v, v] ← 0                                // base case
6
7  For k ← 1 to |V|:
8      For i ← 1 to |V|:
9          For j ← 1 to |V|:
10             // if new path is shorter, reduce distance
11             If dist[i, j] > dist[i, k] + dist[k, j]:
12                 dist[i, j] ← dist[i, k] + dist[k, j]
13
14 Return dist
```

```python
def floyd_warshall(G):
  # let G be an adjacency matrix
  dist = G

  for k in range(len(G)):
    for i in range(len(G)):
      for j in range(len(G)):
        if dist[i][j] > dist[i][k] + dist[k][j]:
          dist[i][j] = dist[i][k] + dist[k][j]

  return dist
```

Indent guides with big main text

indent-guides: 1pt + black

main-text-styles: x => text(size: 15pt)[#x]

FLOYD-WARSHALL(V, E, w):

```
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                    // edge weights
For v in V:
    dist[v, v] ← 0                          // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

```
def floyd warshall(G):
    # let G be an adiacencv matrix
    dist = G


    for k in range(len(G)):
        for
```

Indent guides with big line numbers

indent-guides: 1pt + black

line-number-styles: x => (size: 15pt)[#x]

```
FLOYD-WARSHALL(V, E, w):
  Let dist[u, v] ← ∞ for u, v in V
  For (u, v) in E:
  |   dist[u, v] ← w(u, v)                    // edge weights
  For v in V:
  |   dist[v, v] ← 0                          // base case

  For k ← 1 to |V|:
  |   For i ← 1 to |V|:
  |   |   For j ← 1 to |V|:
  |   |   |   // if new path is shorter, reduce distance
  |   |   |   If dist[i, j] > dist[i, k] + dist[k, j]:
  |   |   |   |   dist[i, j] ← dist[i, k] + dist[k, j]

  Return dist
```

```python
def floyd_warshall(G):
    # let G be an adjacency matrix
    dist = G

    for k in range(len(G)):
        for i in range(len(G)):
            for j in range(len(G)):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
```

`algo` indent guides with big comments

indent-guides: 1pt + black
comment-styles: x => text(size: 15pt, x)

```
FLOYD-WARSHALL(V, E, w):
1  Let dist[u, v] ← ∞ for u, v in V
2  For (u, v) in E:
3  |   dist[u, v] ← w(u, v)                          // edge weights
4  For v in V:
5  |   dist[v, v] ← 0                                // base case
6
7  For k ← 1 to |V|:
8  |   For i ← 1 to |V|:
9  |   |   For j ← 1 to |V|:
10 |   |   |   // if new path is shorter, reduce distance
11 |   |   |   If dist[i, j] > dist[i, k] + dist[k, j]:
12 |   |   |   |   dist[i, j] ← dist[i, k] + dist[k, j]
13
14 Return dist
```

Alignment

indent-guides: 1pt + black

block-align: bottom + right

FLOYD-WARSHALL(V, E, w):

```
1  Let dist[u, v] ← ∞ for u, v in V
2  For (u, v) in E:
3  |   dist[u, v] ← w(u, v)                    // edge weights
4  For v in V:
5  |   dist[v, v] ← 0                          // base case
6
7  For k ← 1 to |V|:
8  |   For i ← 1 to |V|:
9  |   |   For j ← 1 to |V|:
10 |   |   |   // if new path is shorter, reduce distance
11 |   |   |   If dist[i, j] > dist[i, k] + dist[k, j]:
12 |   |   |   |   dist[i, j] ← dist[i, k] + dist[k, j]
13
14 Return dist
```

```python
def floyd_warshall(G):
    # let G be an adjacency matrix
    dist = G

    for k in range(len(G)):
        for i in range(len(G)):
            for j in range(len(G)):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
```

Breakable

indent-guides: 1pt + black
breakable: true

FLOYD-WARSHALL(V, E, w):
1 **Let** $\text{dist}[u, v] \leftarrow \infty$ **for** u, v **in** V
2 **For** (u, v) **in** E**:**
3 | $\text{dist}[u, v] \leftarrow w(u, v)$ // edge weights
4 **For** v **in** V**:**
5 | $\text{dist}[v, v] \leftarrow 0$ // base case
6
7 **For** $k \leftarrow 1$ **to** $|V|$**:**
8 | **For** $i \leftarrow 1$ **to** $|V|$**:**
9 | | **For** $j \leftarrow 1$ **to** $|V|$**:**

```
        // if new path is shorter, reduce distance
        If dist[i, j] > dist[i, k] + dist[k, j]:
            dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

```python
def floyd_warshall(G):
    # let G be an adjacency matrix
    dist = G

    for k in range(len(G)):
        for i in range(len(G)):
```

```
            for j in range(len(G)):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

        return dist
```

Broken indent guides with small inset

row-gutter:	15pt
inset:	3pt
indent-guides:	1pt + black
breakable:	true

```
FLOYD-WARSHALL(V, E, w):
    Let dist[u, v] ← ∞ for u, v in V
    For (u, v) in E:
        dist[u, v] ← w(u, v)                    // edge weights
    For v in V:
        dist[v, v] ← 0                          // base case

    For k ← 1 to |V|:
        For i ← 1 to |V|:
            For j ← 1 to |V|:
```

```
        // if new path is shorter, reduce distance
        If dist[i, j] > dist[i, k] + dist[k, j]:
            dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

```python
def floyd_warshall(G):
    # let G be an adjacency matrix
    dist = G

    for k in range(len(G)):
        for i in range(len(G)):
            for j in range(len(G)):
```

```
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
```

Custom keyword/comment styling, conditional line numbering

keyword-styles:
```
x => if x in ([for], [in], [to], [:]) {
  text(blue, x)
} else {
  underline(text(blue, x))
}
```

line-number-styles:
```
i => if calc.rem(i, 5) != 0 {
  text(gray, 8pt)[#i]
} else [#i]
```

FLOYD-WARSHALL(V, E, w):

```
Let dist[u, v] ← ∞ for u, v in V
For (u, v) in E:
    dist[u, v] ← w(u, v)                    // edge weights
For v in V:
    dist[v, v] ← 0                          // base case

For k ← 1 to |V|:
    For i ← 1 to |V|:
        For j ← 1 to |V|:
            // if new path is shorter, reduce distance
            If dist[i, j] > dist[i, k] + dist[k, j]:
                dist[i, j] ← dist[i, k] + dist[k, j]

Return dist
```

```python
def floyd_warshall(G):
  # let G be an adjacency matrix
  dist = G

  for k in range(len(G)):
    for i in range(len(G)):
      for j in range(len(G)):
        if dist[i][j] > dist[i][k] + dist[k][j]:
          dist[i][j] = dist[i][k] + dist[k][j]

  return dist
```

Algo Keyword Regex Test

keywords: `("A", "B\*", "\*C", "\*D\*", "\+", "E\*E", "FF?F", "G{4}")`

keyword-styles: `x => text(red, x)`

Regex Tester():

```
1  A AA AAA A* *A *A* A*A
2  A+ +A +A+ A+A
3  B BB BBB B* *B *B* B*B
4  B+ +B +B+ B+B
5  C CC CCC C* *C *C* C*C
6  C+ +C +C+ C+C
7  D DD DDD D* *D *D* D*D
8  D+ +D +D+ D+D
9  E EE EEE E* *E *E* E*E
10 E+ +E +E+ E+E
11 F FF FFF F* *F *F* F*F
12 F+ +F +F+ F+F
13 G GG GGG GGGG GGGGG GGGGGG
14 H HH HHH HHHH HHHHH HHHHHH
15 if let if iif if letter
16 while let while wwhile while letter
```