**Floyd-Warshall FW Algorithm**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

The Floyd-Warshall algorithm is a dynamic programming approach used to find the shortest paths between all pairs of vertices in a weighted graph. It works on both directed and undirected graphs and efficiently handles negative edge weights, making it suitable for graphs with negative cycles. By iteratively considering all intermediate vertices as potential shortcuts between two vertices, the algorithm efficiently computes the shortest distances between all pairs of vertices, resulting in a matrix containing the shortest paths.

**Use cases:**

The Floyd-Warshall algorithm efficiently computes the shortest paths between all pairs of vertices in a single pass, distinguishing it from other algorithms that focus on single-source or single-destination paths. By employing dynamic programming and considering intermediate vertices as potential shortcuts, it creates a matrix with shortest distances, enabling quick retrieval of shortest paths for any pair of vertices in a weighted graph. This makes it a powerful choice for scenarios requiring all-pairs shortest path calculations.

**Algorithm:**

1. # Variables

2. inf = float('inf')

3.

4. #APSP Floyd-Warshall Algorithm

5. def fw(graph):

6.     n = len(graph)

7.     r = [[float('inf') if i != j else 0 for j in range(n)] for i in range(n)]

8.

9.     for i in range(n):

10.         for j in range(n):

11.             if not graph[i][j] == 0:

12.                 r[i][j] = graph[i][j]

13.     # The Actual Algorithm

14.     for k in range(n):

15.         for i in range(n):

16.             for j in range(n):

17.                 r[i][j] = min(r[i][k] + r[k][j], r[i][j])

18.

19.     # Detect negative cycles

20.     for k in range(n):

21.         for i in range(n):

22.             for j in range(n):

23.                 if r[i][j] > r[i][k] + r[k][j]:

24.                     r[i][j] = -inf

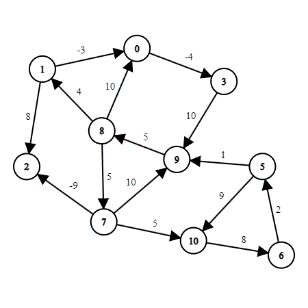
25.     return r

26.

27. show(fw(graph))

**Example:**

Here’s a small example illustrating an example of input outputs for the Floyd-Warshall FW:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. # Variables

2. inf = float('inf')

3. graph = [

4.     #  0    1    2    3    4    5    6    7   8   9     10

5.     [  0,   0,   0,  -4,   0,   0,   0,   0,  0,   0,   0], # 0

6.     [ -3,   0,   8,   0,   0,   0,   0,   0,  0,   0,   0], # 1

7.     [  0,   0,   0,   0,   0,   0,   0,   0,  0,   0,   0], # 2

8.     [  0,   0,   0,   0,   0,   0,   0,   0,  0,  10,   0], # 3

9.     [  0,   0,   0,   0,   0,   0,   0,   0,  0,   0,   0], # 4

10.     [  0,   0,   0,   0,   0,   0,   0,   0,  0,   1,   9], # 5

11.     [  0,   0,   0,   0,   0,   2,   0,   0,  0,   0,   0], # 6

12.     [  0,   0,  -9,   0,   0,   0,   0,   0,  0,  10,   5], # 7

13.     [ 10,   4,   0,   0,   0,   0,   0,   5,  0,   0,   0], # 8

14.     [  0,   0,   0,   0,   0,   0,   0,   0,  5,   0,   0], # 9

15.     [  0,   0,   0,   0,   0,   0,   8,   0,  0,   0,   0], # 10

16. ]

17.

18. #APSP Floyd-Warshall Algorithm

19. def fw(graph):

20.     n = len(graph)

21.     r = [[float('inf') if i != j else 0 for j in range(n)] for i in range(n)]

22.

23.     for i in range(n):

24.         for j in range(n):

25.             if not graph[i][j] == 0:

26.                 r[i][j] = graph[i][j]

27.     # The Actual Algorithm

28.     for k in range(n):

29.         for i in range(n):

30.             for j in range(n):

31.                 r[i][j] = min(r[i][k] + r[k][j], r[i][j])

32.

33.     # Detect negative cycles

34.     for k in range(n):

35.         for i in range(n):

36.             for j in range(n):

37.                 if r[i][j] > r[i][k] + r[k][j]:

38.                     r[i][j] = -inf

39.     return r

40.

41. show(fw(graph))

42.

The corresponding output is:

1. Python>> 00 15 07 -4 +∞ 31 29 16 11 06 21

2. >> -3 00 04 -7 +∞ 28 26 13 08 03 18

3. >> +∞ +∞ 00 +∞ +∞ +∞ +∞ +∞ +∞ +∞ +∞

4. >> 16 19 11 00 +∞ 35 33 20 15 10 25

5. >> +∞ +∞ +∞ +∞ 00 +∞ +∞ +∞ +∞ +∞ +∞

6. >> 07 10 02 03 +∞ 00 17 11 06 01 09

7. >> 09 12 04 05 +∞ 02 00 13 08 03 11

8. >> 16 19 -9 12 +∞ 15 13 00 15 10 05

9. >> 01 04 -4 -3 +∞ 20 18 05 00 07 10

10. >> 06 09 01 02 +∞ 25 23 10 05 00 15

11. >> 17 20 12 13 +∞ 10 08 21 16 11 00

