Floyd-Warshall Algorithm

Dynamic Programing

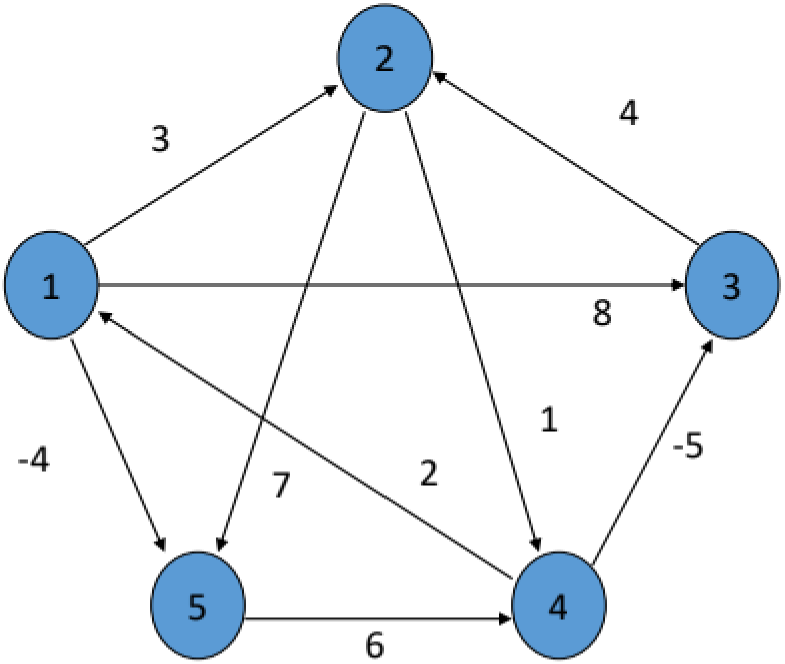
Floyd-Warshall Algorithm

- It is an application of **Dynamic Programming**.
- The algorithm finds **all-pairs shortest paths** on a graph i.e., it is guaranteed to find the shortest path between every pair of vertices in a graph.
- The graph may have **negative weight edges**, but no negative weight cycles (for then the shortest path is undefined).

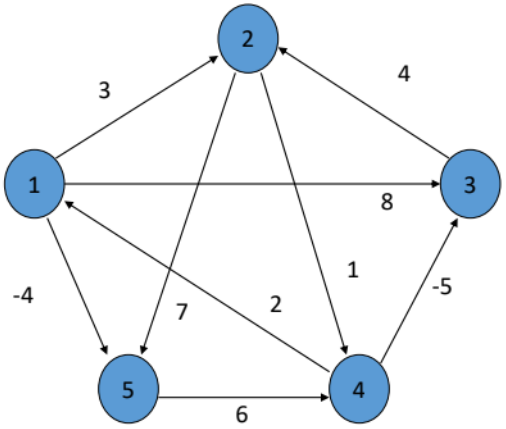
Algorithm

```
• n = no of vertices
• Initialize $dist^0$ matrix of dimension n*n
• for k = 1 to n
•   for i = 1 to n
•     for j = 1 to n
•       if $dist^{k-1}[i,j] > dist^{k-1}[i,k] + dist^{k-1}[k,j]$
•         $dist^k[i,j] = dist^{k-1}[i,k] + dist^{k-1}[k,j]$
• return A
```

Example



- Create matrices $dist(0)$



dist(0)	1	2	3	4	5
1	0	3	8	INF	-4
2	INF	0	INF	1	7
3	INF	4	0	INF	INF
4	2	INF	-5	0	INF
5	INF	INF	INF	6	0

- Create $dist(1)$ from $dist(0)$
- $(k = 1)$ Freeze elements in 1st row and 1st column
- Remaining rows and column
    - If $dist[i][j] > dist[i][1] + dist[1][j]$
        - Then $dist[i][j] = dist[i][1] + dist[1][j]$

dist(0)	1	2	3	4	5
1	0	3	8	INF	-4
2	INF	0	INF	1	7
3	INF	4	0	INF	INF
4	2	INF	-5	0	INF
5	INF	INF	INF	6	0

dist(1)	1	2	3	4	5
1	0	3	8	INF	-4
2	INF	0	INF	1	7
3	INF	4	0	INF	INF
4	2	5	-5	0	-2
5	INF	INF	INF	6	0

- Create $dist(2)$ from $dist(1)$
- $(k = 2)$ Freeze elements in 2nd row and column
- Remaining rows and column
  - If $dist[i][j] > dist[i][2] + dist[2][j]$
    - Then $dist[i][j] = dist[i][2] + dist[2][j]$

dist(1)	1	2	3	4	5
1	0	3	8	INF	-4
2	INF	0	INF	1	7
3	INF	4	0	INF	INF
4	2	5	-5	0	-2
5	INF	INF	INF	6	0

dist(2)	1	2	3	4	5
1	0	3	8	4	-4
2	INF	0	INF	1	7
3	INF	4	0	5	11
4	2	5	-5	0	-2
5	INF	INF	INF	6	0

- Create $dist(3)$ from $dist(2)$
- $(k = 3)$ Freeze elements in 3rd row and column
- Remaining rows and column
  - If $dist[i][j] > dist[i][3] + dist[3][j]$
    - Then $dist[i][j] = dist[i][3] + dist[3][j]$

dist(2)	1	2	3	4	5
1	0	3	8	4	-4
2	INF	0	INF	1	7
3	INF	4	0	5	11
4	2	5	-5	0	-2
5	INF	INF	INF	6	0

dist(3)	1	2	3	4	5
1	0	3	8	4	-4
2	INF	0	INF	1	7
3	INF	4	0	5	11
4	2	-1	-5	0	-2
5	INF	INF	INF	6	0

- Create $dist(4)$ from $dist(3)$
- $(k = 4)$ Freeze elements in 4th row and column
- Remaining rows and column
  - If $dist[i][j] > dist[i][4] + dist[4][j]$
    - Then $dist[i][j] = dist[i][4] + dist[4][j]$

dist(3)	1	2	3	4	5
1	0	3	8	4	-4
2	INF	0	INF	1	7
3	INF	4	0	5	11
4	2	-1	-5	0	-2
5	INF	INF	INF	6	0

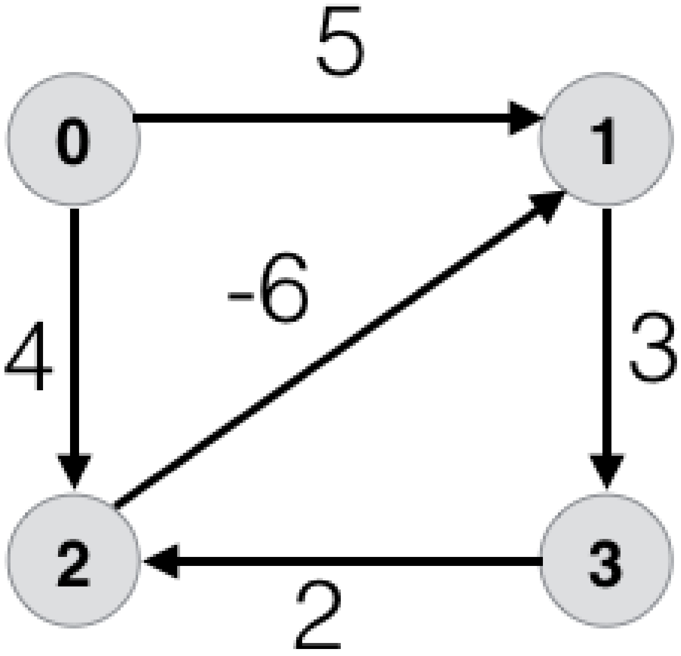
dist(4)	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

- Create $dist(5)$ from $dist(4)$
- $(k = 5)$ Freeze elements in 5th row and column
- Remaining rows and column
  - If $dist[i][j] > dist[i][5] + dist[5][j]$
    - Then $dist[i][j] = dist[i][5] + dist[5][j]$

dist(4)	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

dist(5)	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

Practice



Negative Cycle

- This algorithm can also be used to **detect** the presence of **negative cycles** (where the sum of the edges in a cycle is negative)
  - At the **end of the algorithm**, the distance from a vertex v to itself is negative ($\boldsymbol{dist(v,v) < 0}$).