Floyd–Warshall algorithm

Group 7 including:

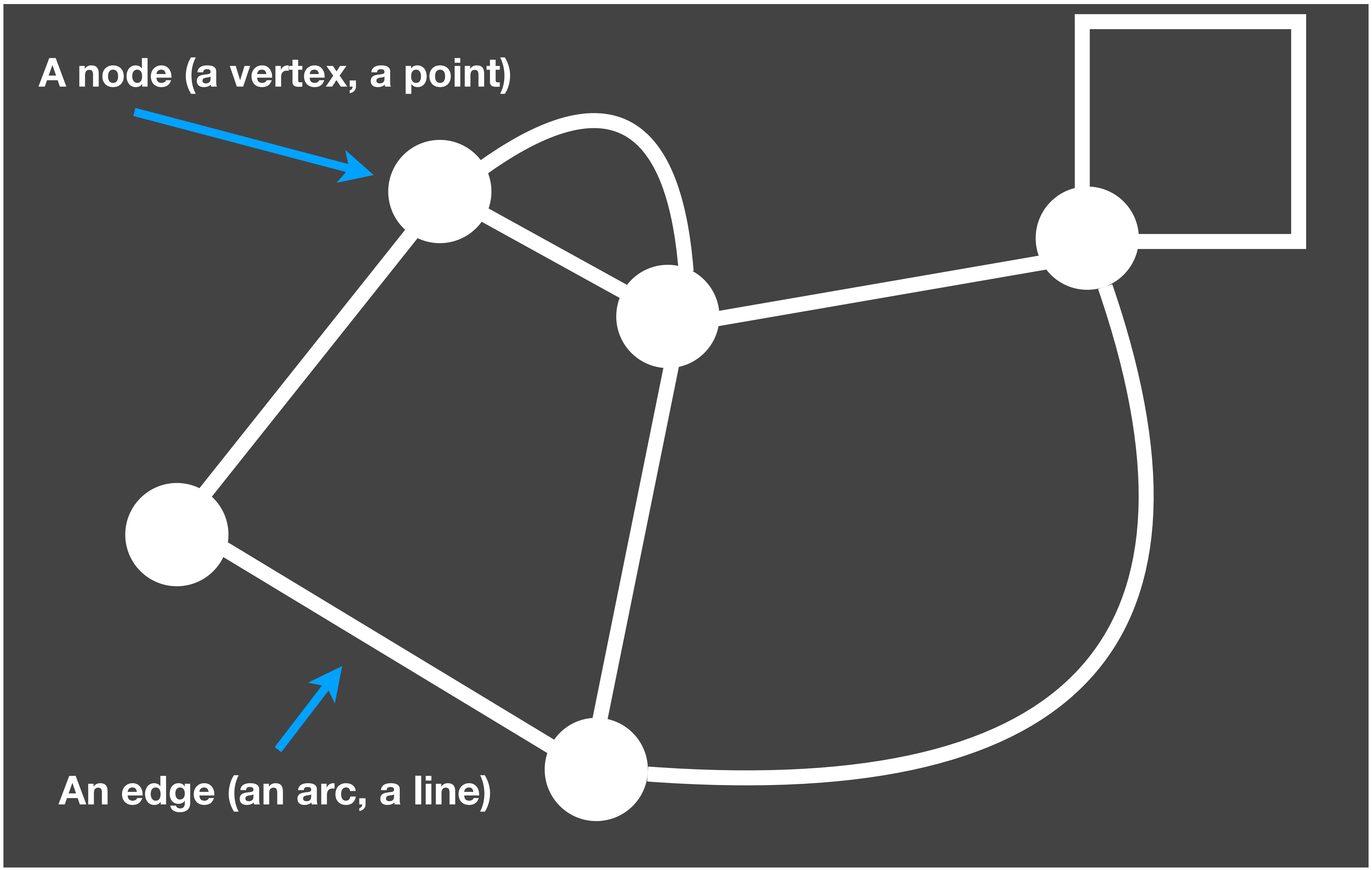
- 1751013 - Nguyen Ho Huu Nghia
- 1751024 - Nguyen Ngoc Phuong Trang
- 1751064 - Nguyen Hoang Gia
- 1751082 - Vu Hoang Minh

Definition

"The Floyd–Warshall algorithm is an algorithm for finding **shortest paths** in a **weighted graph** with positive or negative edge weights (but with no negative cycles)"
- Wikipedia.org -

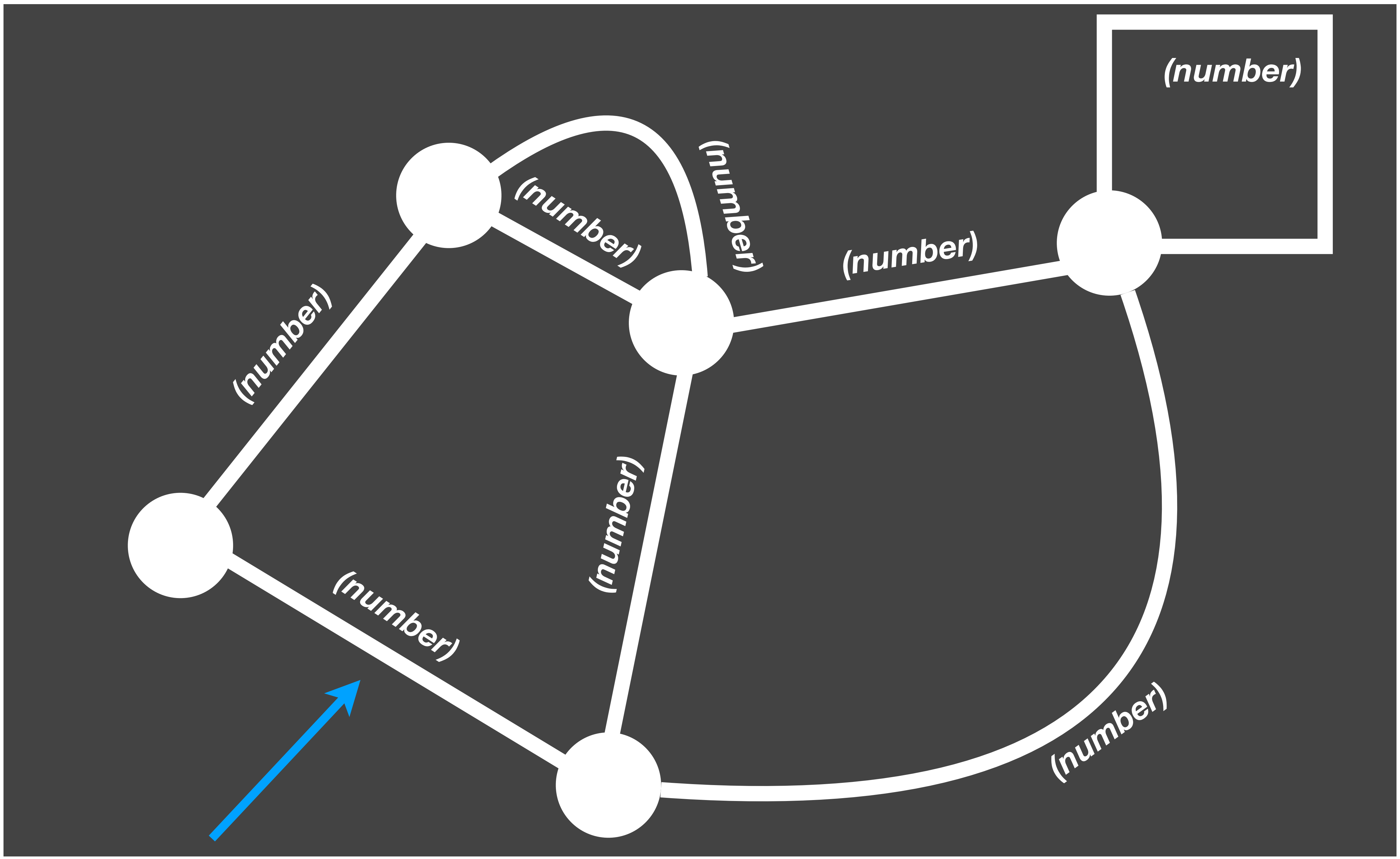
Definition

What is "Graph"?



Definition

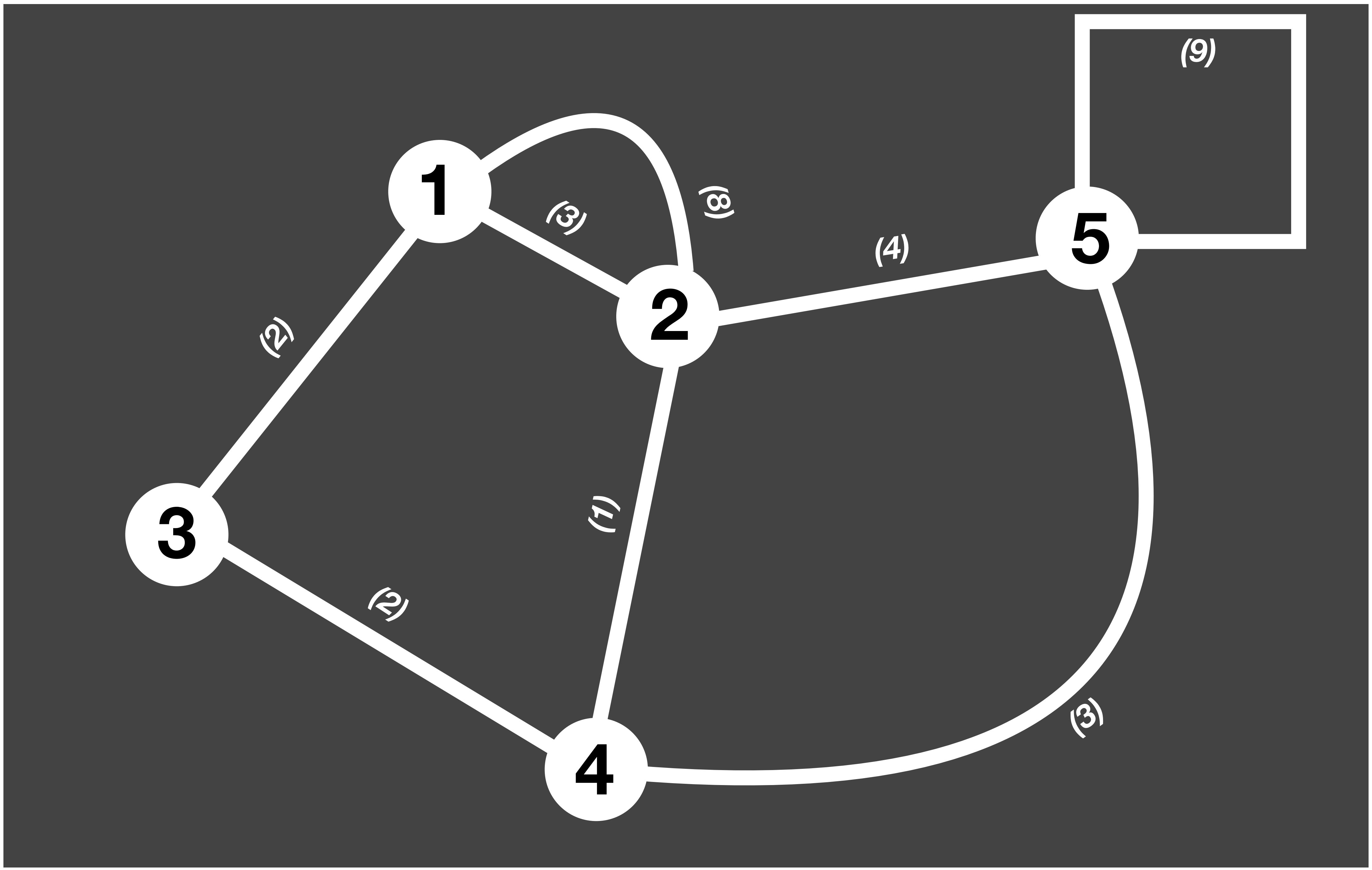
What is "Weighted Graph"?



Simply just edges contain numbers. These numbers might represent for costs, lengths, capacities, e.t.c... which are depending on the problem you face.

Definition

An example for "Weighted Graph"



Floyd–Warshall algorithm

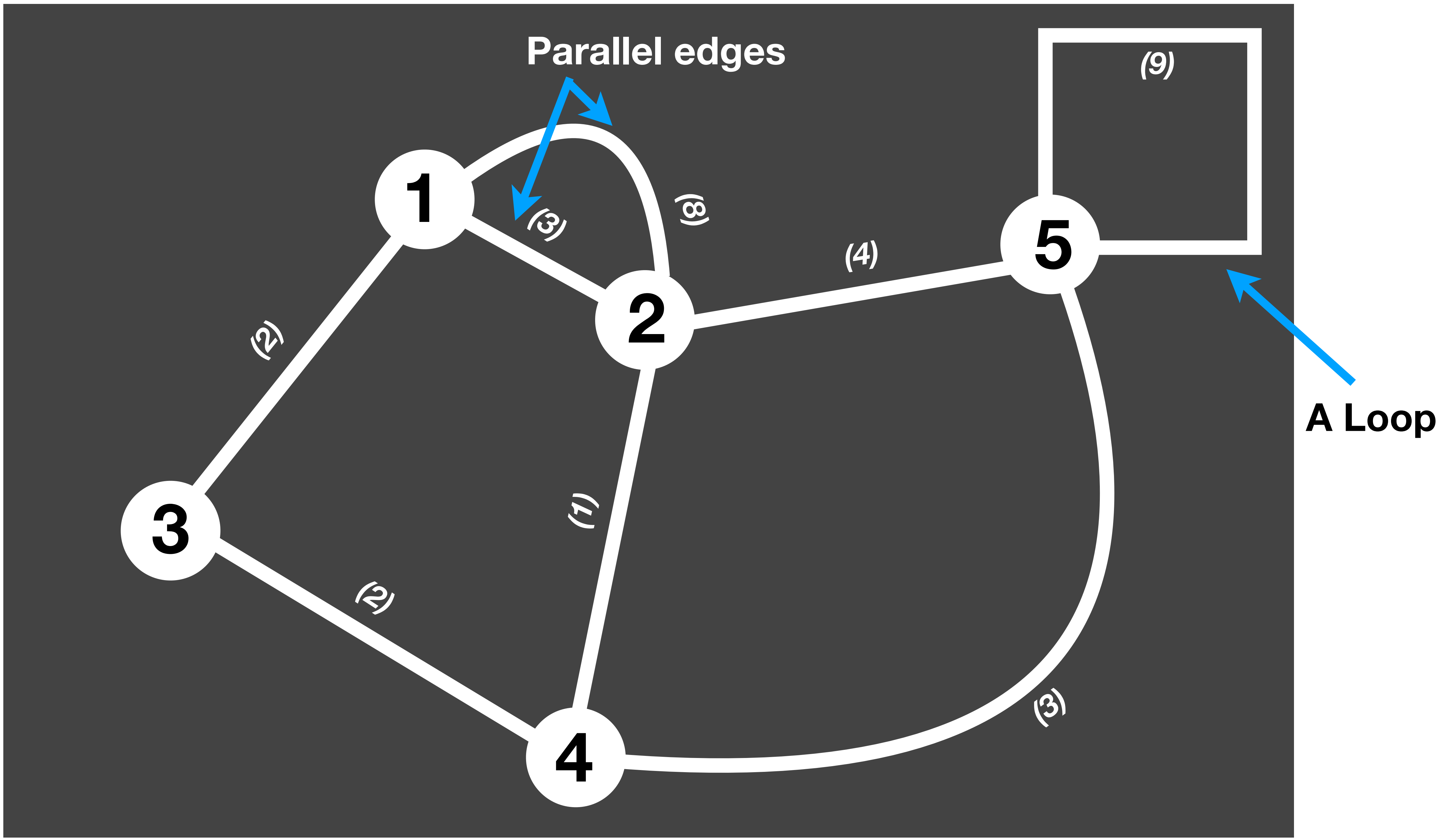
How does it work?

Step 1: Check the graph whether it is "valid" or not. This including remove:

- Loops (Edge starts and ends at the same node)
- Parallel edges (Remove all edges between 2 nodes and keep the edge with the smallest weight)

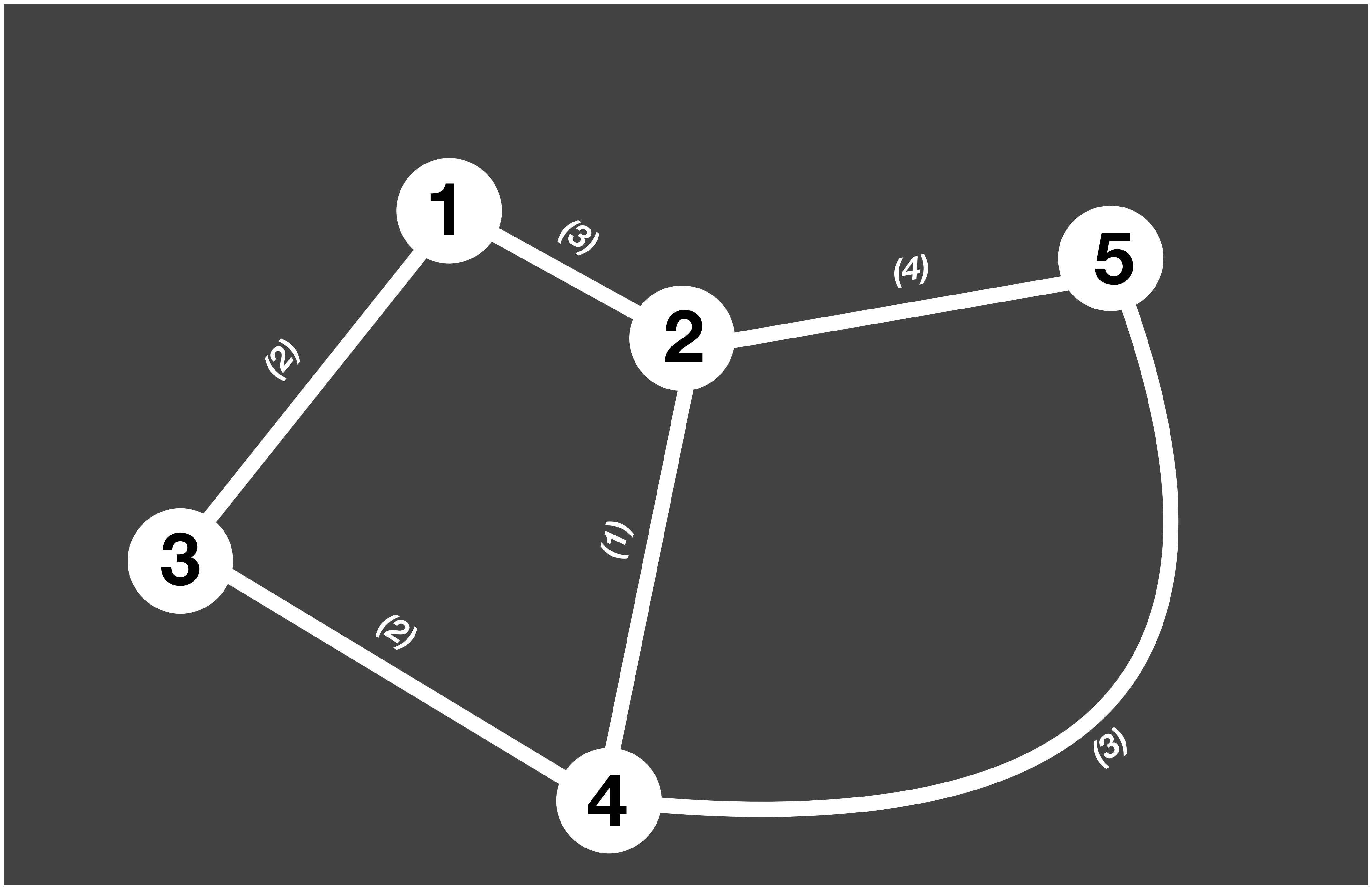
Floyd–Warshall algorithm

How does it work?



Floyd–Warshall algorithm

How does it work?



Floyd–Warshall algorithm

How does it work?

Step 2: We will visual representation for this algorithm by drawing 2 matrices called: Distance Matrix (D) and Sequence Matrix (S).

(D) will keep the value of distance between any two nodes.
(S) will keep the node's name that helps finding the shortest part between any two nodes.

If the graph has n node(s) then (D) & (S) will be **n x n**.

Floyd–Warshall algorithm

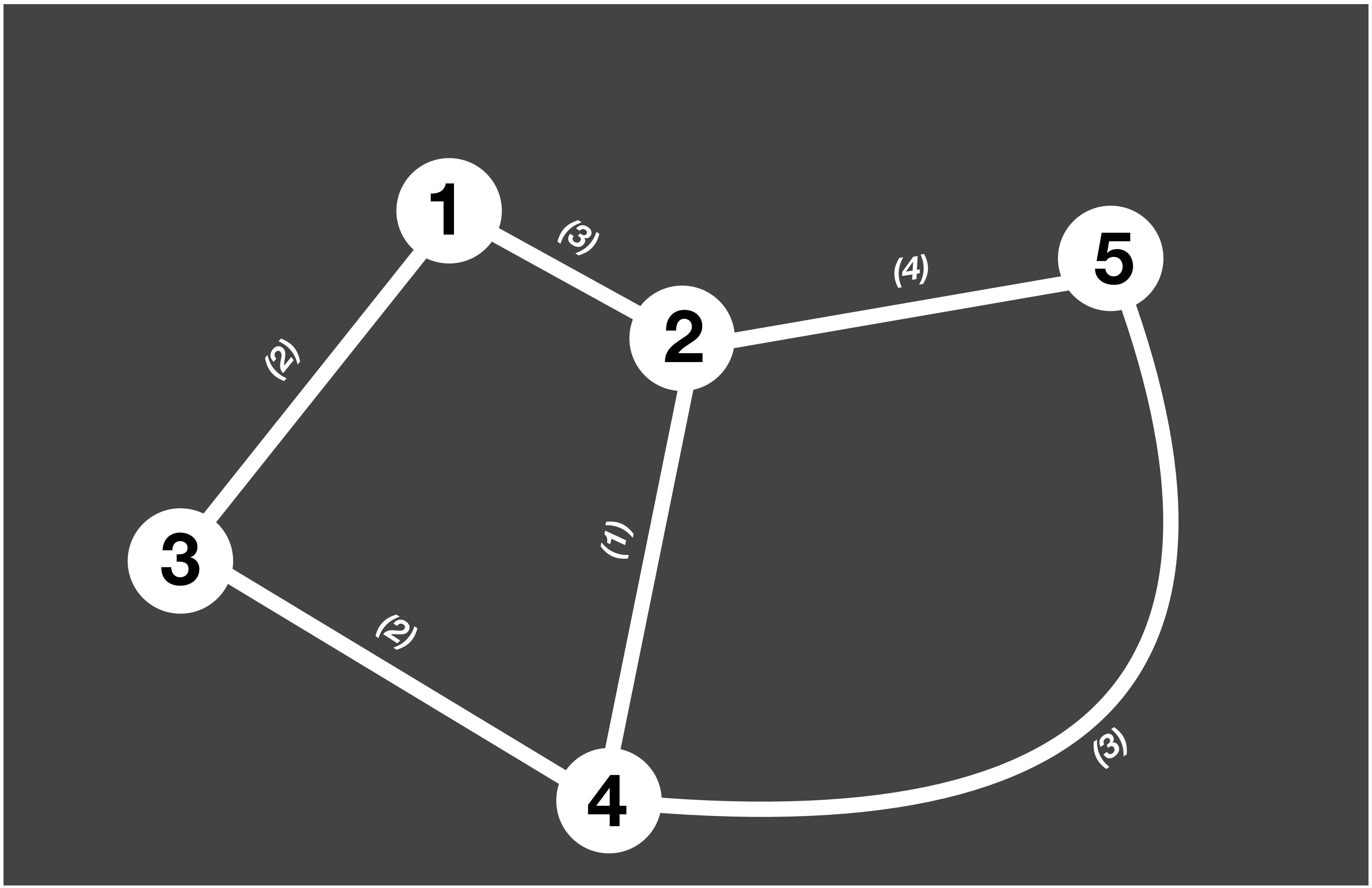
How does it work?

Consider some notations:

- k: iteration number.
- D(k): Distance matrix in k-th iteration.
- S(k): Sequence matrix in k-th iteration.
- D(i,j): Distance between 2 node **i** and **j**.

Floyd–Warshall algorithm

How does it work?

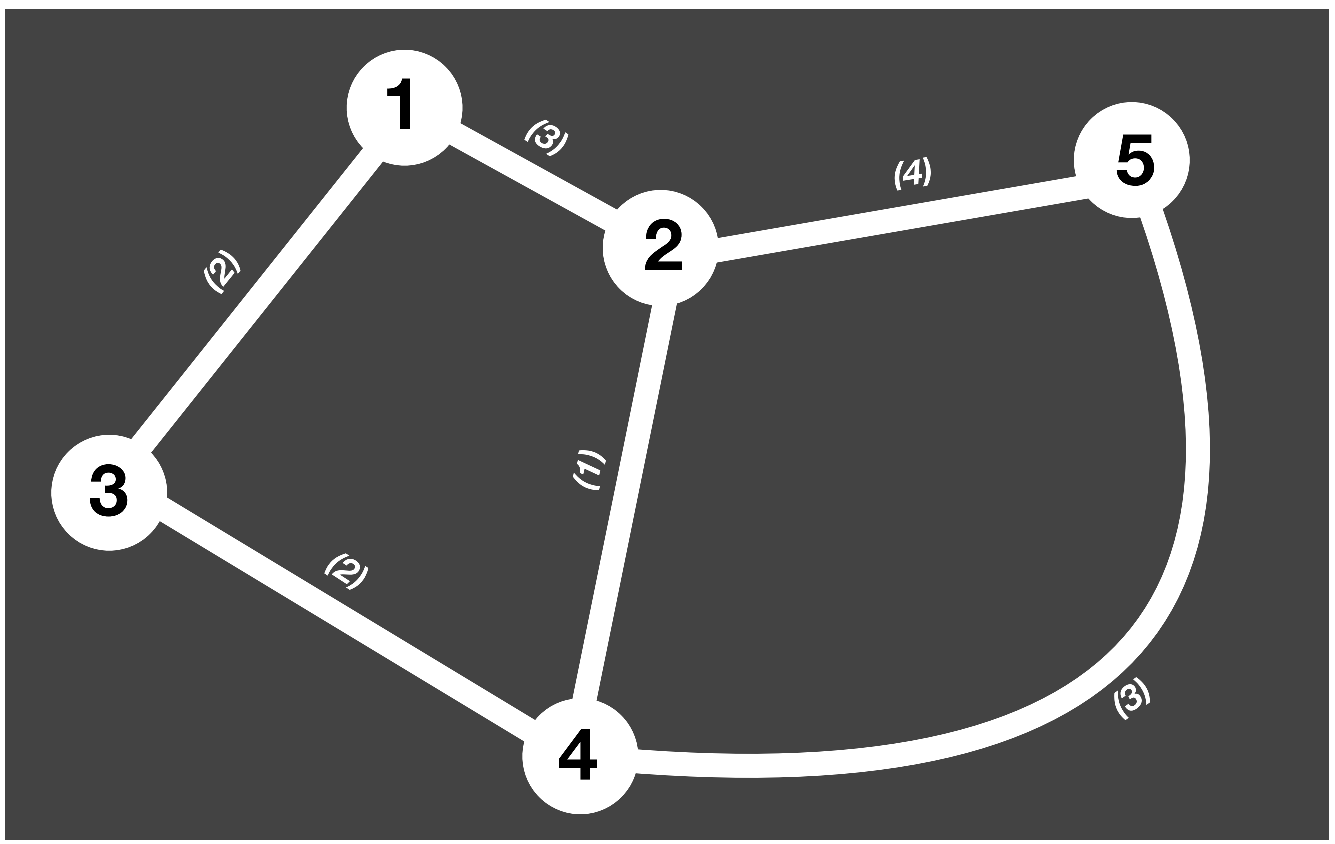


Floyd–Warshall algorithm

How does it work?

k = 0

D(0)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-



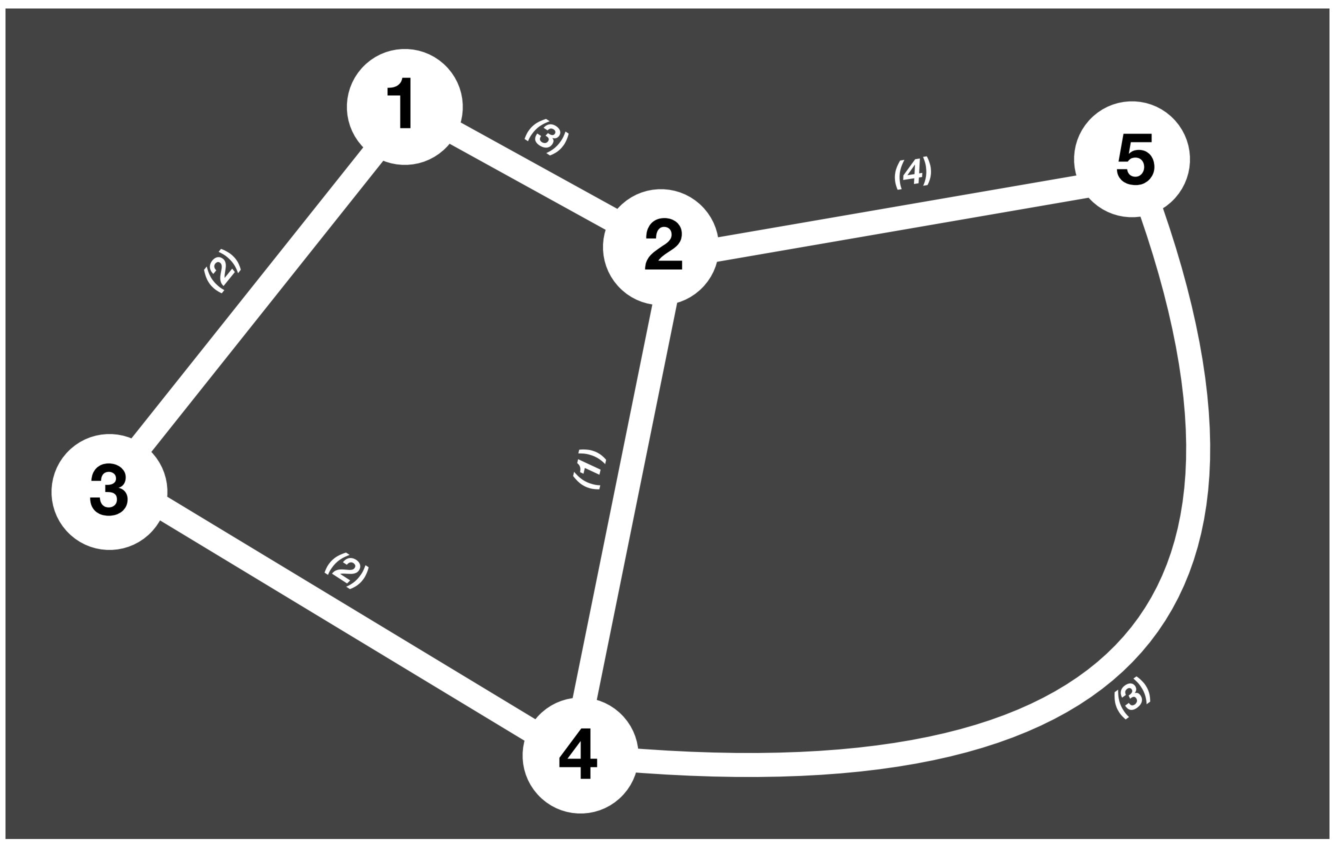
After step 1, we have no longer conflict with loops and parallel edges. Hence, (D) & (S) with same position for row and column will be NULL.
E.g.: [1,1]; [2,2]; ...

S(0)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

Floyd–Warshall algorithm

How does it work?

k = 0



D(0)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

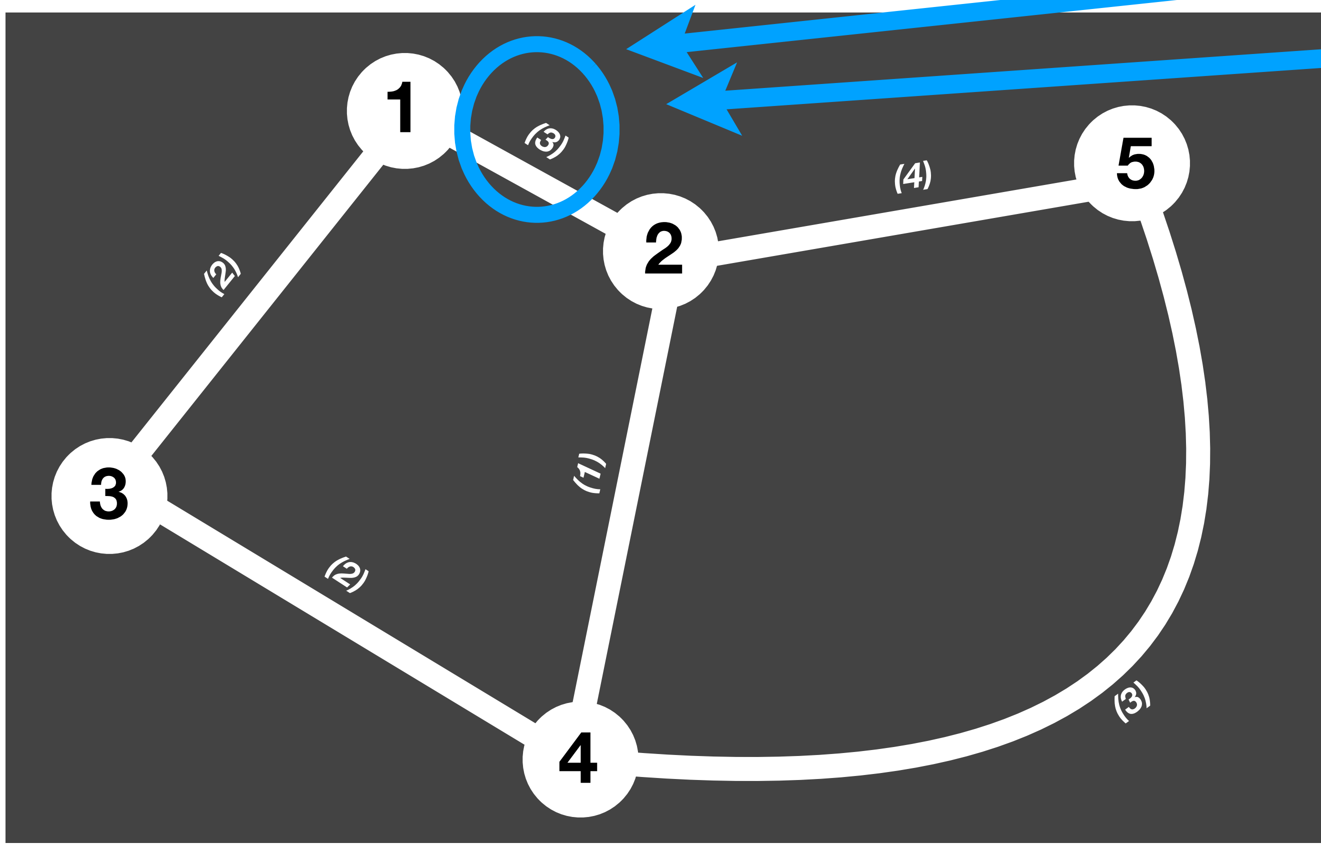
S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Now, every cell in (S) will be matched with its column.

Floyd–Warshall algorithm

How does it work?

k = 0



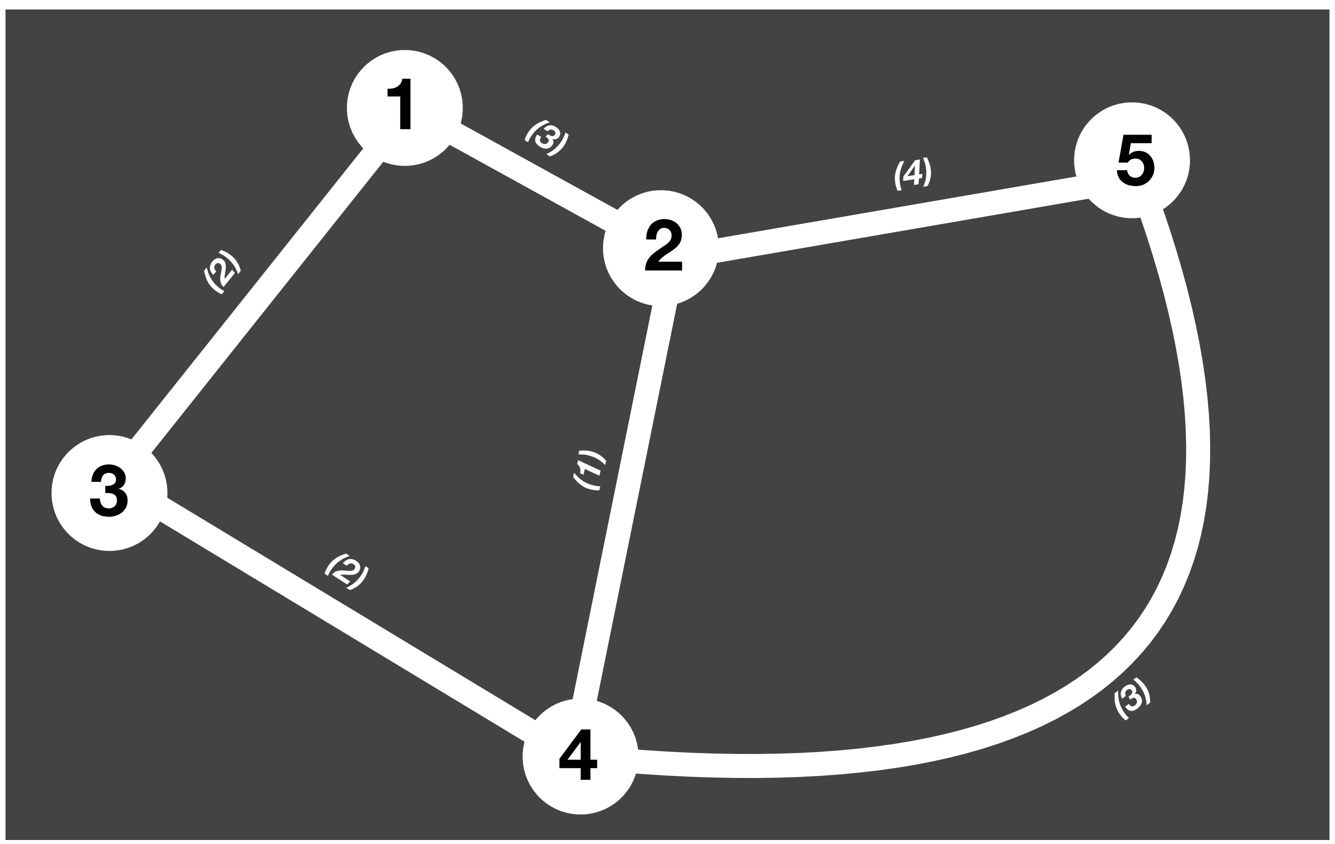
D(0)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?

k = 0



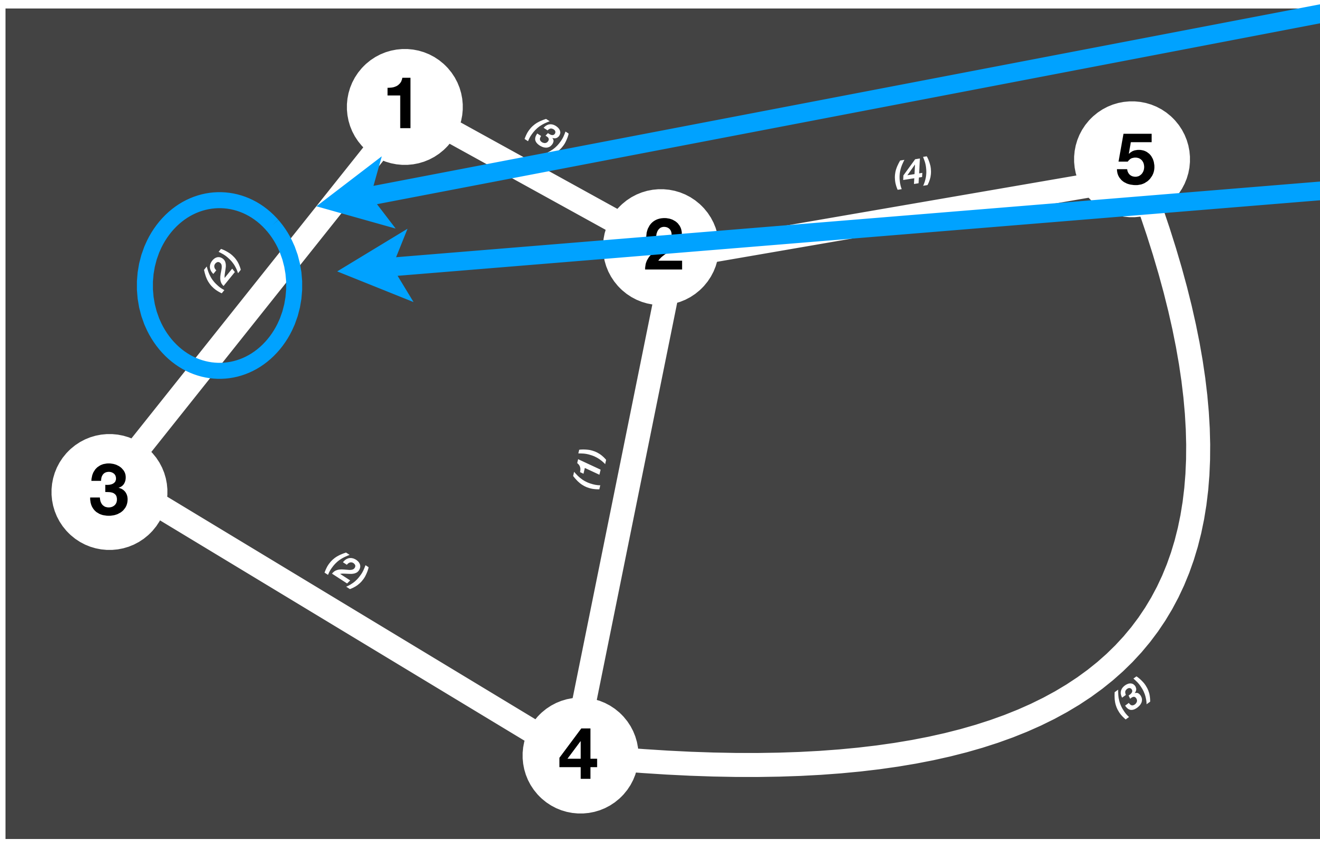
D(0)	1	2	3	4	5
1	-	3			
2	3	-			
3			-		
4				-	
5					-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?

k = 0



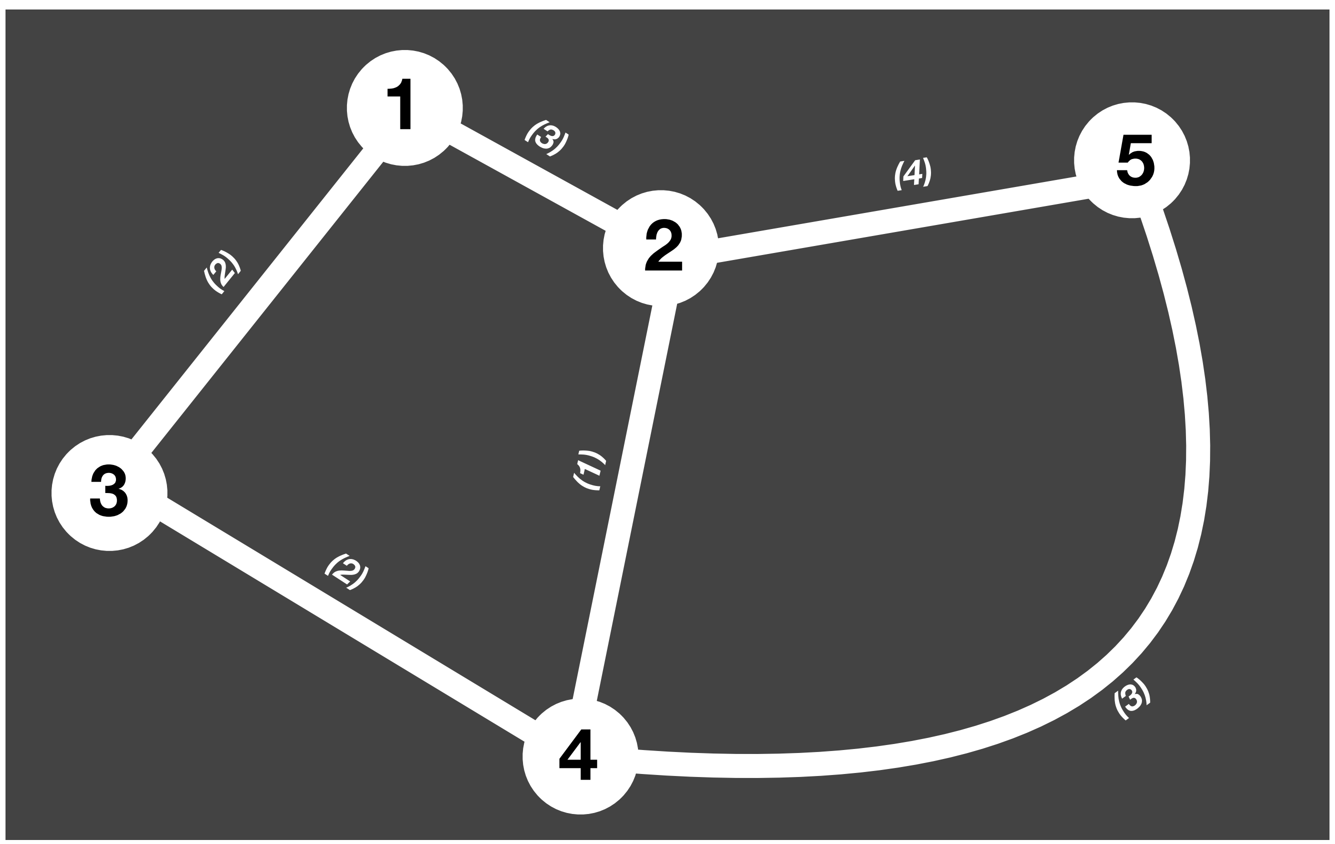
D(0)	1	2	3	4	5
1	-	3			
2	3	-			
3			-		
4				-	
5					-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?

k = 0



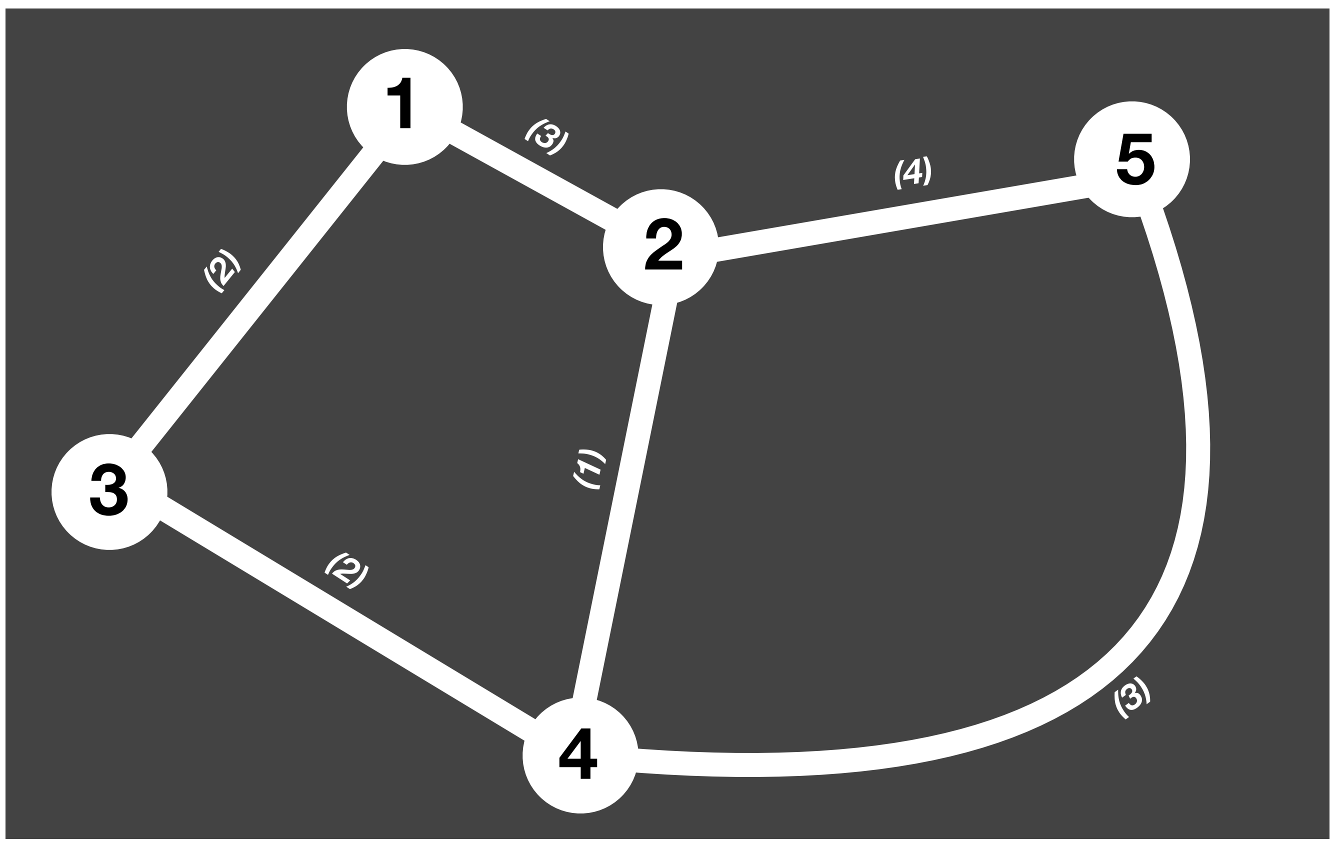
D(0)	1	2	3	4	5
1	-	3	2		
2	3	-			
3	2		-		
4				-	
5					-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?

k = 0



D(0)	1	2	3	4	5
1	-	3	2		
2	3	-		1	4
3	2		-	2	
4		1	2	-	3
5		4		3	-

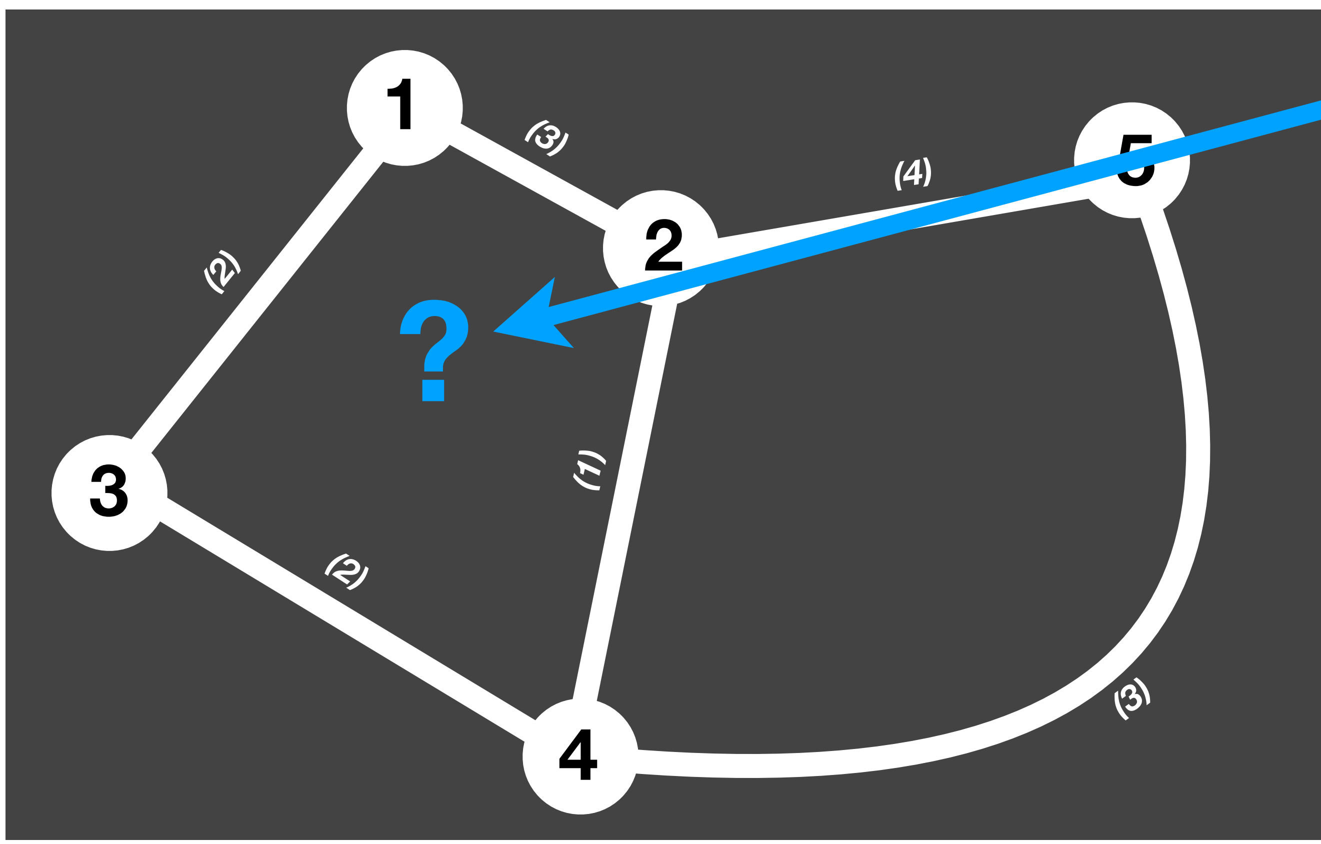
S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?

k = 0

D(0)	1	2	3	4	5
1	-	3	2		
2	3	-		1	4
3	2		-	2	
4		1	2	-	3
5		4		3	-



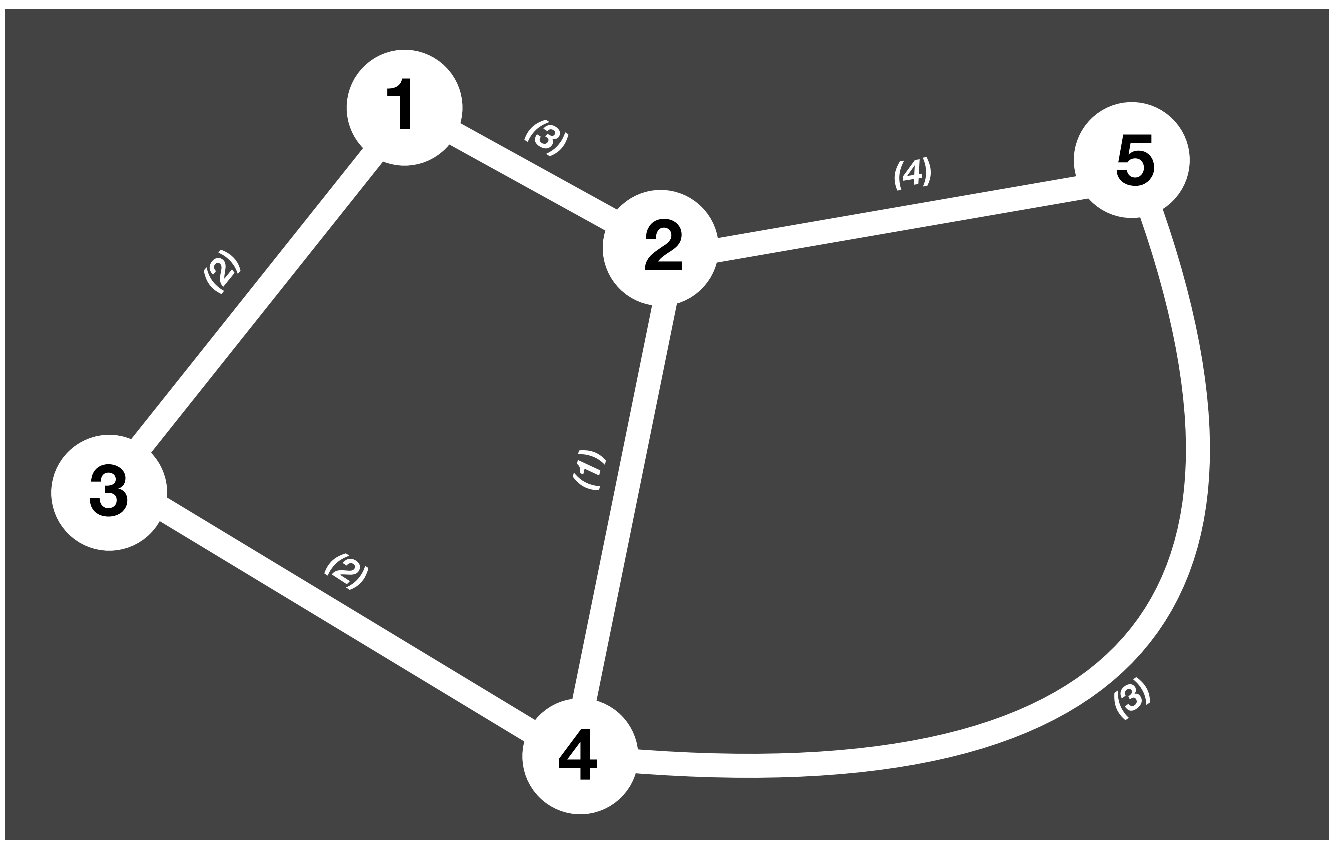
S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

2 nodes are connected with no edge, we will consider the distance between them is **infinity** (∞)

Floyd–Warshall algorithm

How does it work?

k = 0



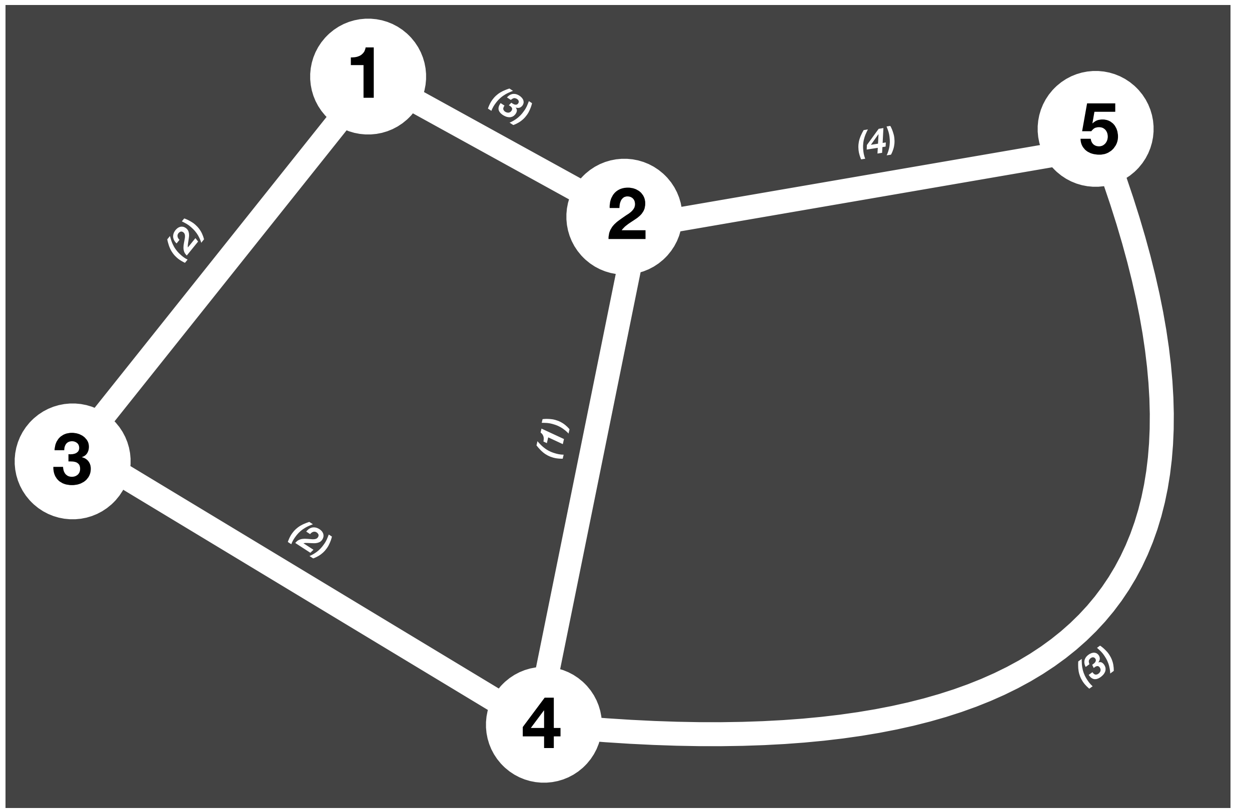
2 nodes are connected with no edge, we will consider the distance between them is **infinity** (∞)

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?



k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

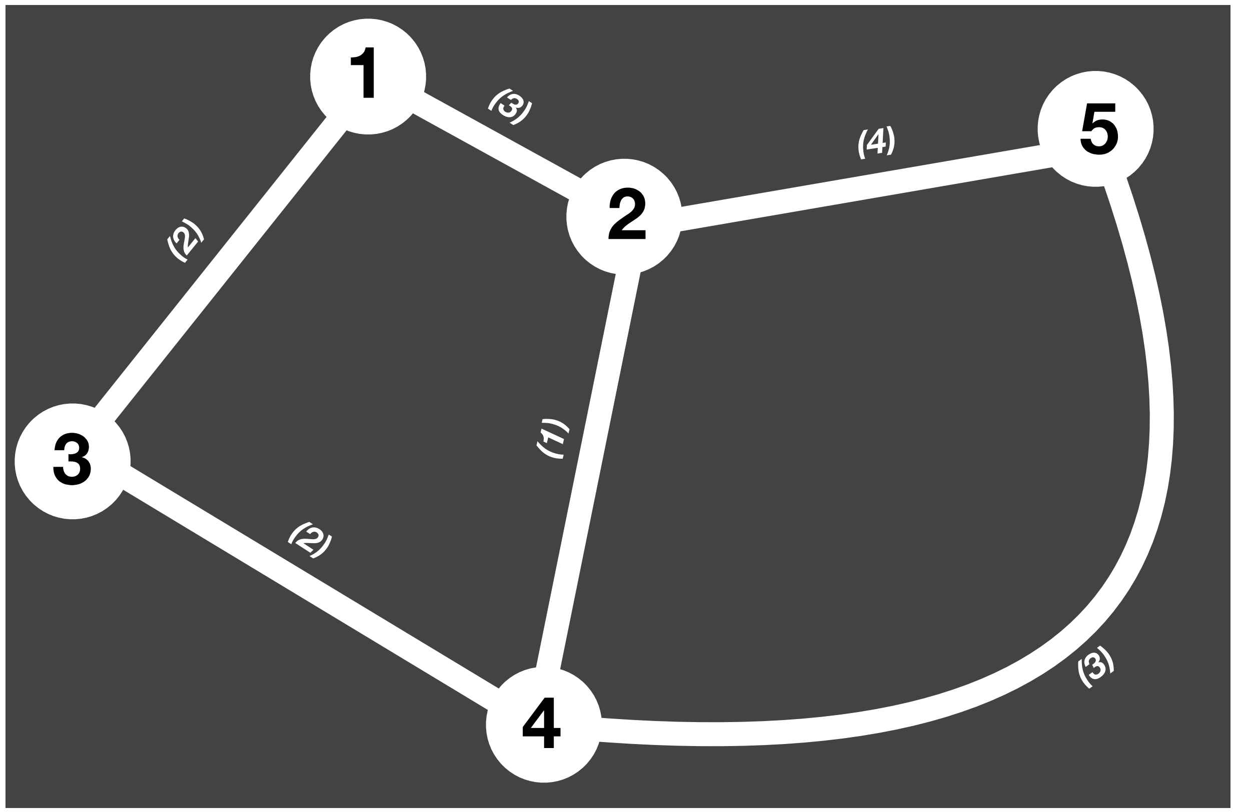
D(1)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

Floyd–Warshall algorithm

How does it work?



k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

When k increases, we build D(k) and S(k) base on previous matrices of k-1.

We will copy the number of column **k** and row **k** from D(k-1) and S(k-1) to the new matrices D(k) and S(k).

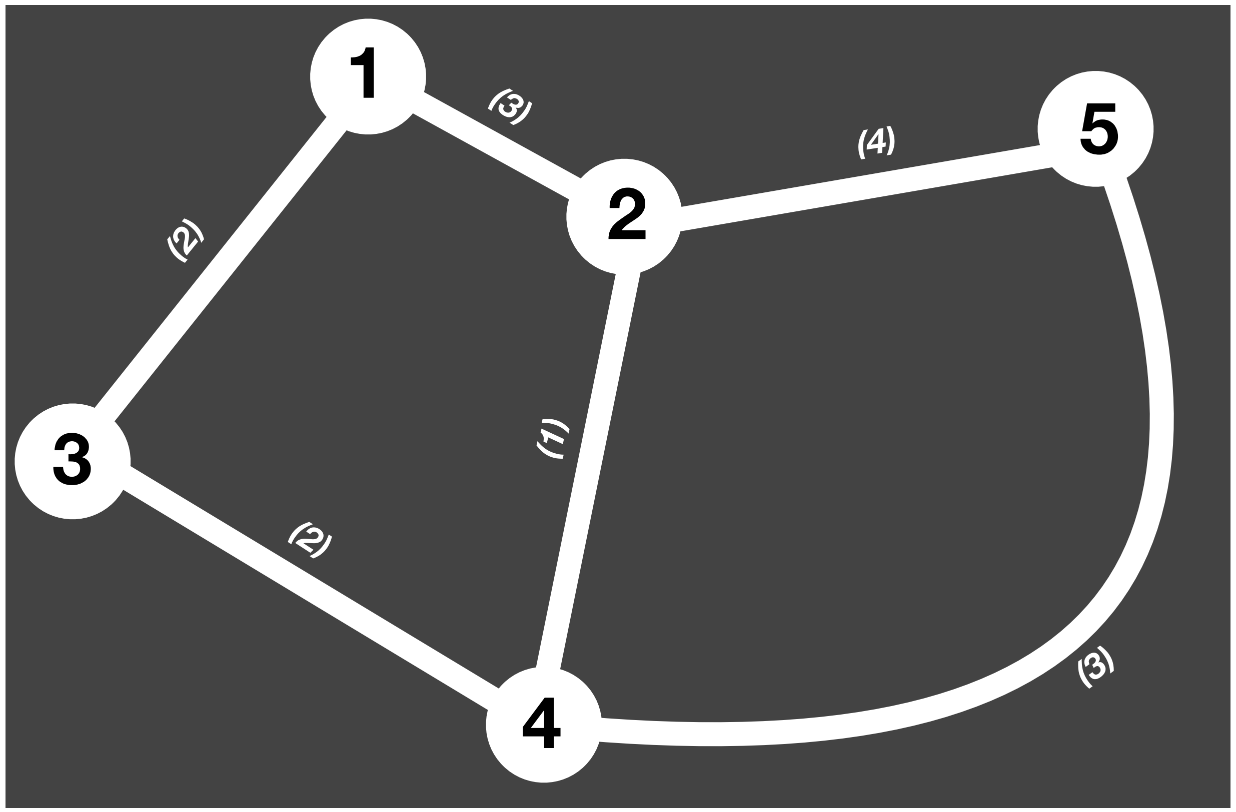
For instance, with k = 1, we'll copy column 1 and row 1 from D(0), S(0) to D(1) & S(1)

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-				
2		-			
3			-		
4				-	
5					-

Floyd–Warshall algorithm

How does it work?



k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-			
3	2		-		
4	∞			-	
5	∞				-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-			
3	1		-		
4	1			-	
5	1				-

When k increases, we build D(k) and S(k) base on previous matrices of k-1.

We will copy the number of column **k** and row **k** from D(k-1) and S(k-1) to the new matrices D(k) and S(k).

For instance, with k = 1, we'll copy column 1 and row 1 from D(0), S(0) to D(1) & S(1)

Floyd–Warshall algorithm

How does it work?

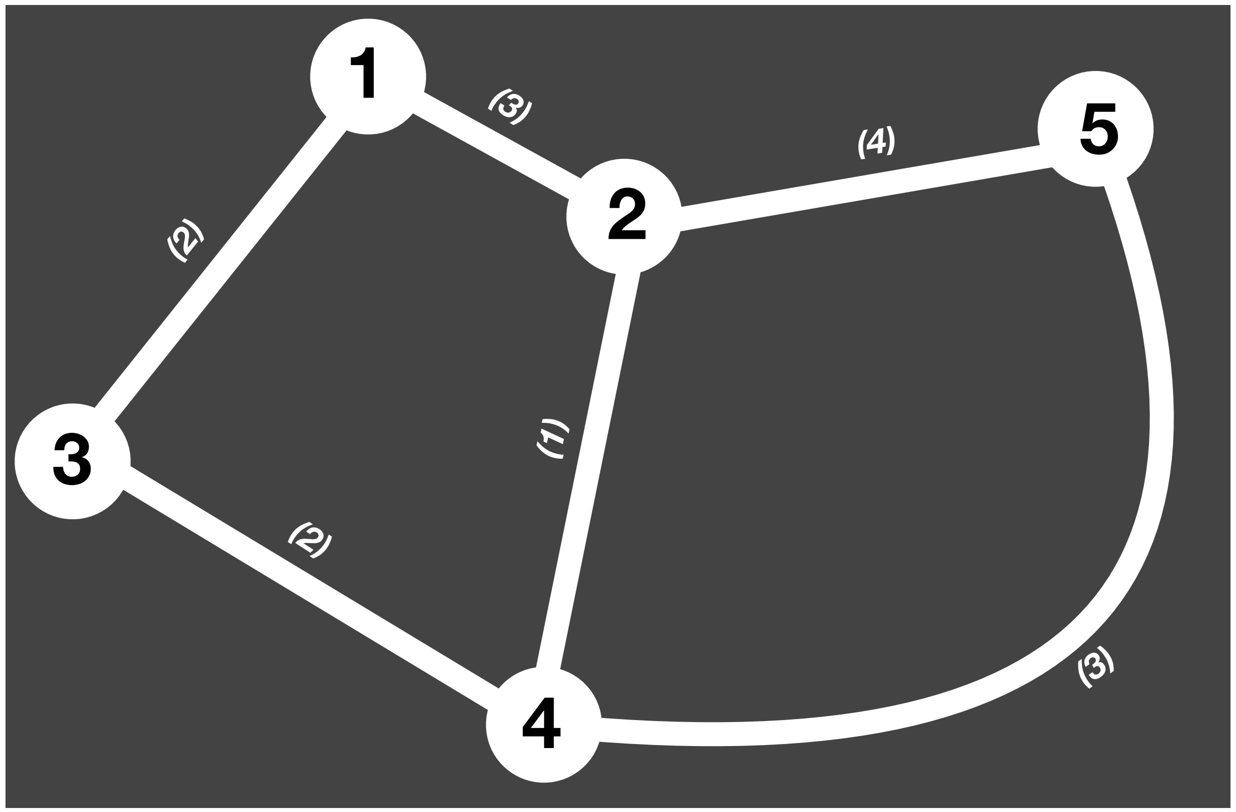
k = 1

After filled, we continue to fill all the cell in matrix D(k), followed by these rules: (Consider the cell will be fill is Cij - with i is the current row and j is the current column.

If **D(i,j) > D(i,k) + D(k,j)** then Cij will be **D(i,k) + D(k,j)** from **D(k-1) matrix,** and the cell in row **i** column **j** of matrix **S(k)** will be **k**. Otherwise, Cij will be **D(i,j)** from **D(k-1)**, and the cell in row **i** column **j** of matrix **S(k)** will be **the same with S(k-1)**.

Floyd–Warshall algorithm

How does it work?



When k increases, we build D(k) and S(k) base on previous matrices of k-1.

We will copy the number of column **k** and row **k** from D(k-1) and S(k-1) to the new matrices D(k) and S(k).

For instance, with k = 1, we'll copy column 1 and row 1 from D(0), S(0) to D(1) & S(1)

k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

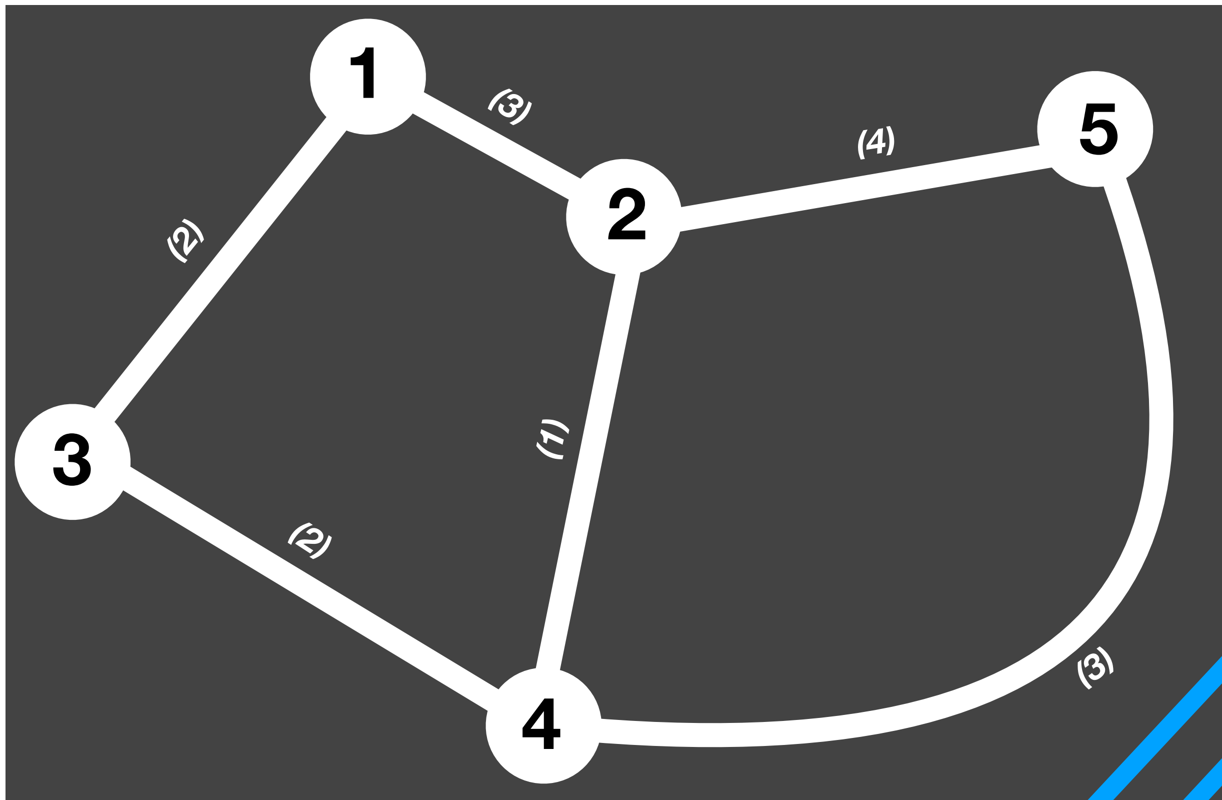
D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-			
3	2		-		
4	∞			-	
5	∞				-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-			
3	1		-		
4	1			-	
5	1				-

Floyd–Warshall algorithm

How does it work?



k = 1

Row 2, Col 3 => C23

D(0)	1	2	3	4	5
1		3	2	∞	∞
2	3		∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-			
3	2		-		
4	∞			-	
5	∞				-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-			
3	1		-		
4	1			-	
5	1				-

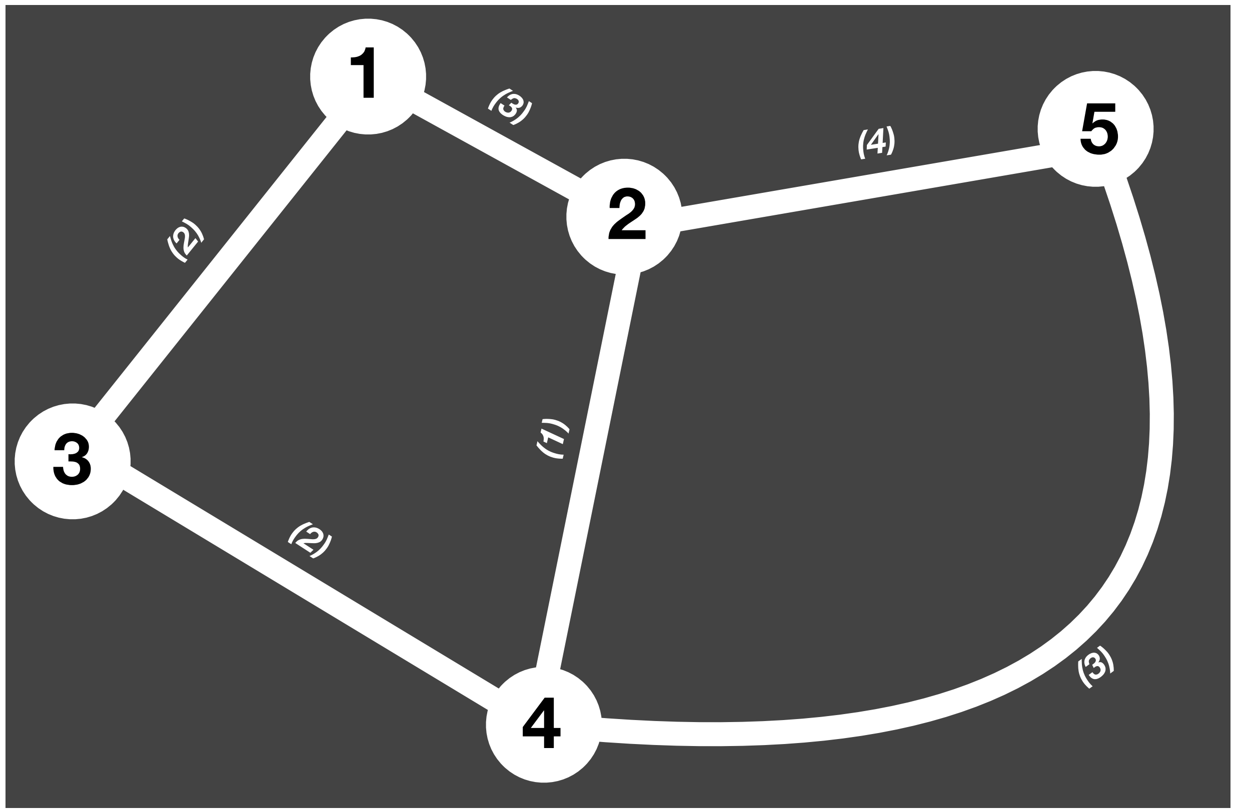
We're currently at C23 so:
i = 2
j = 3

D(i,k) will be D(2,1)
D(k,j) will be D(1,3)

D(2,1) + D(1,3) compare with D(2,3)

Floyd–Warshall algorithm

How does it work?



k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5		
3	2		-		
4	∞			-	
5	∞				-

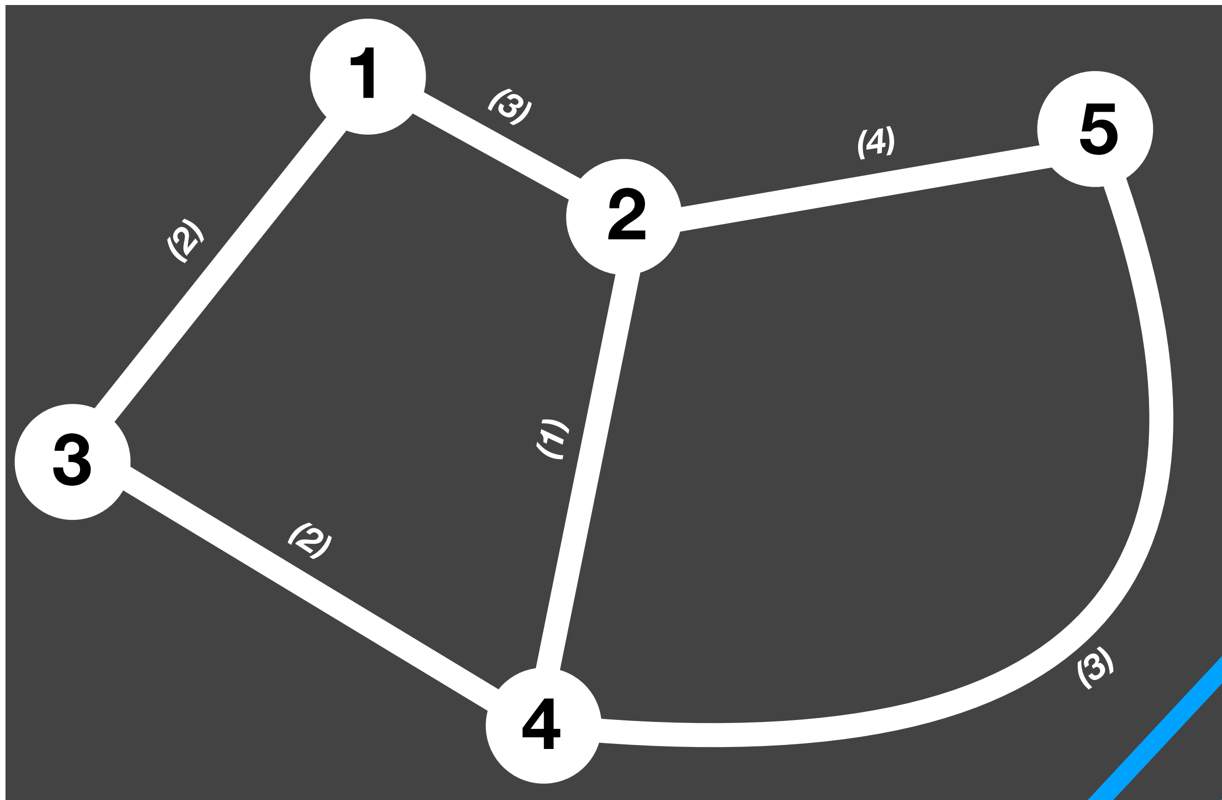
5 > infinity
So cell C23 at D(1) will be **5.**
C23 of S(1) will be k = 1.

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	1		
3	1		-		
4	1			-	
5	1				-

Floyd–Warshall algorithm

How does it work?



k = 1

Row 2, Col 4 => C24

D(0)	1	2	3	4	5
1		3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5		
3	2		-		
4	∞			-	
5	∞				-

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-			
3	1		-		
4	1			-	
5	1				-

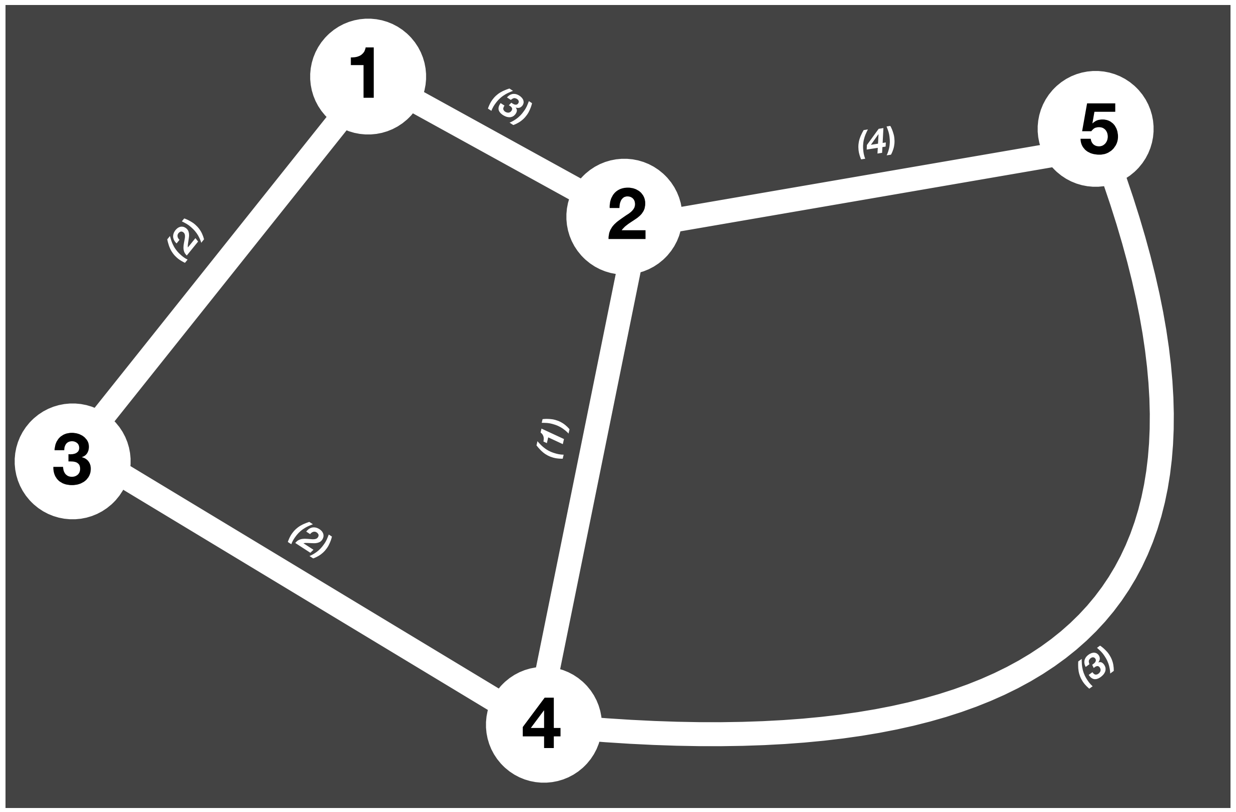
We're currently at C24 so:
i = 2
j = 4

D(i,k) will be D(2,1)
D(k,j) will be D(1,4)

D(2,1) + D(1,4) compare with D(2,4)

Floyd–Warshall algorithm

How does it work?



k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5	1	4
3	2	5	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

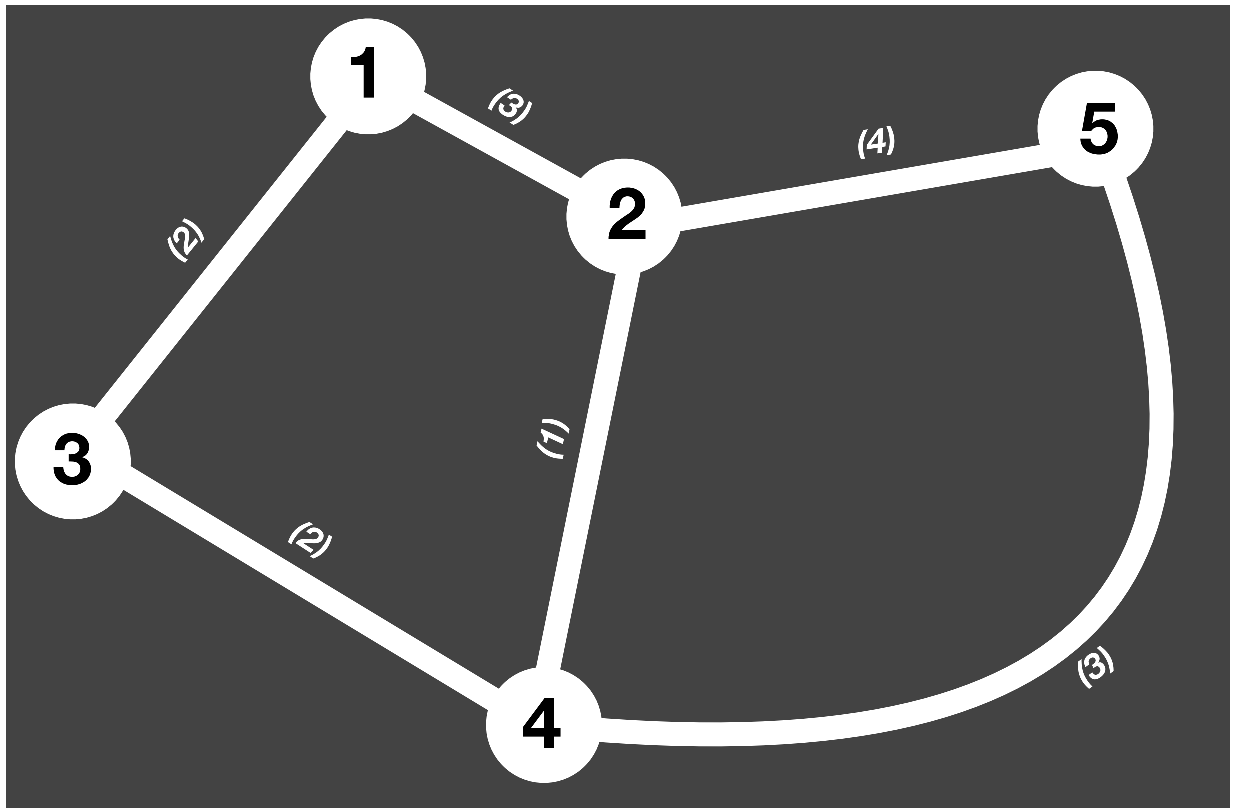
3 + infinity > 1
So cell C24 at D(1) will be **1.**
C24 at S(1) will be the same at
C24 at S(0).

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	1	4	
3	1		-		
4	1			-	
5	1				-

Floyd–Warshall algorithm

How does it work?



k = 1

D(0)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	∞	1	4
3	2	∞	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5	1	4
3	2	5	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

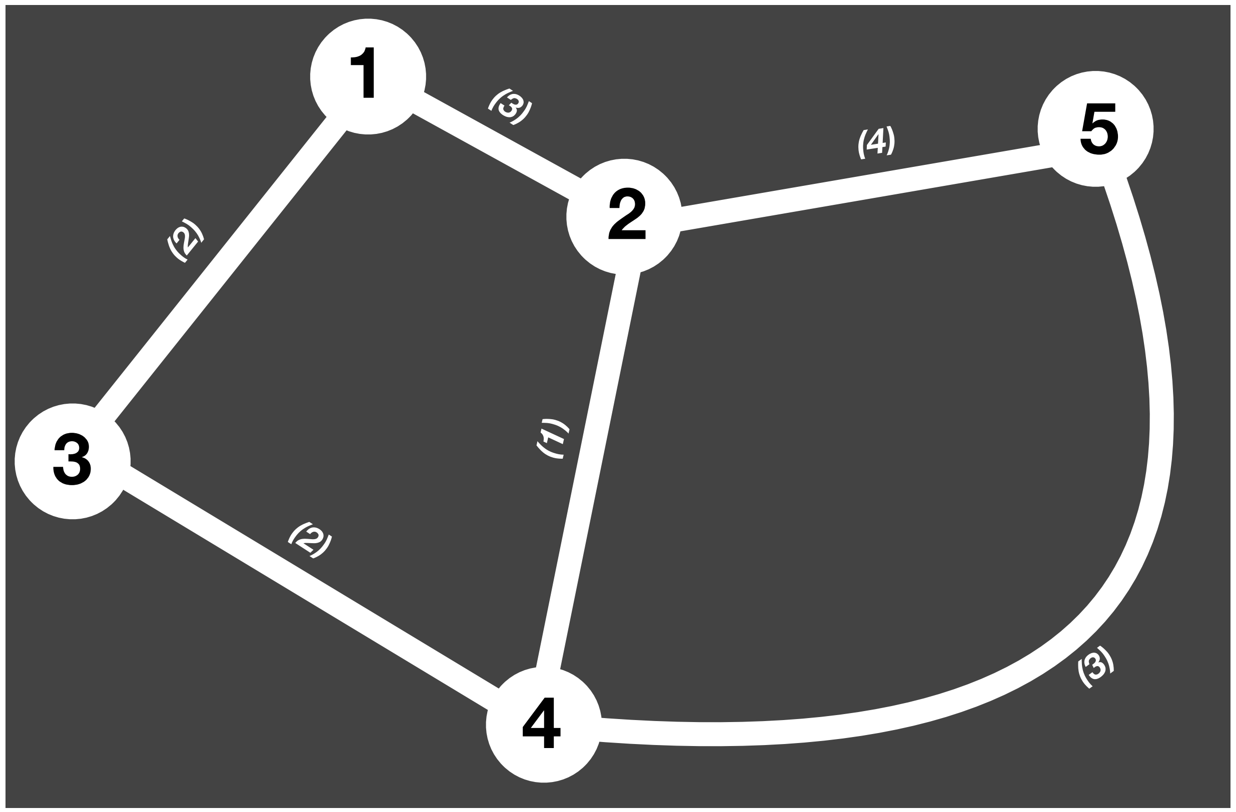
Fill all the blank in D(1) and S(1) with same rules.

S(0)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	3	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	1	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

Floyd–Warshall algorithm

How does it work?



k = 2

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5	1	4
3	2	5	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(2)	1	2	3	4	5
1	-	3			
2	3	-	5	1	4
3		5	-		
4		1		-	
5		4			-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	1	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

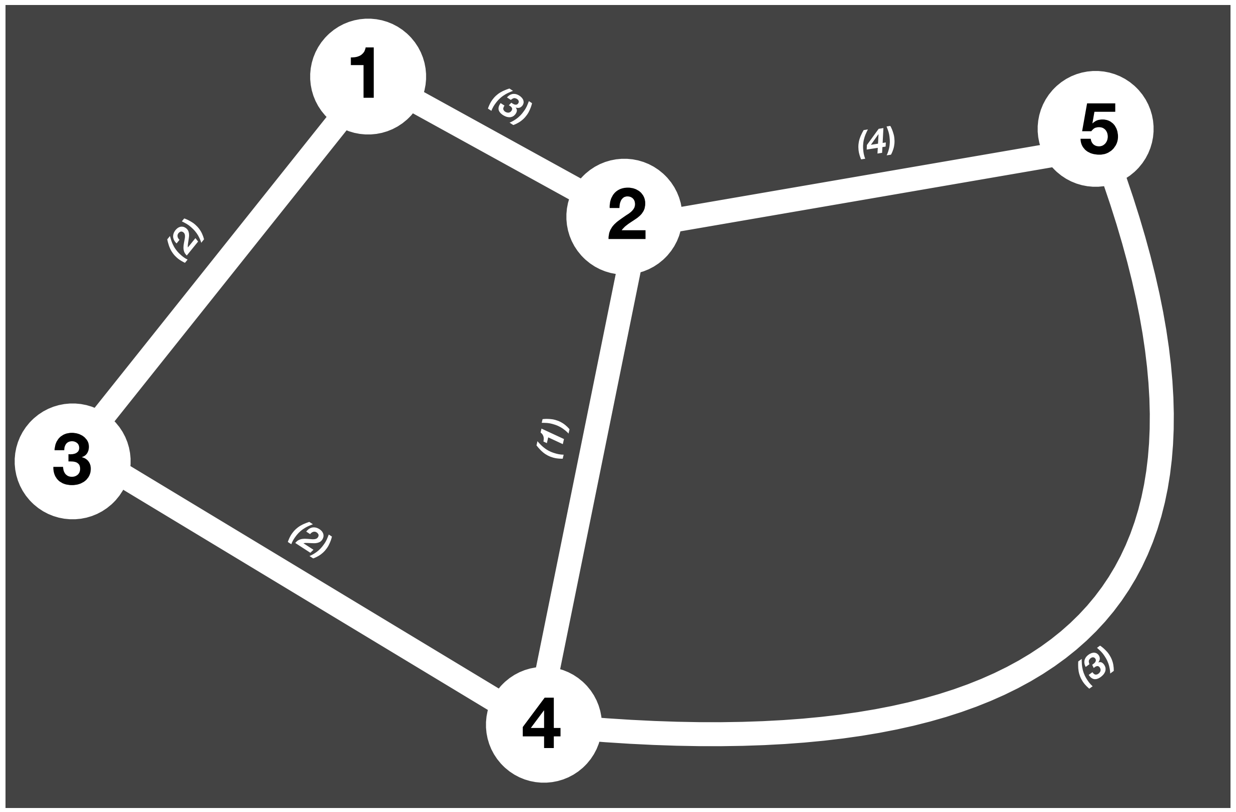
S(2)	1	2	3	4	5
1	-	2			
2	1	-	1	4	5
3		2	-		
4		2		-	
5		2			-

When k increases, we got k = 2.

With k = 2, we'll copy column 2 and row 2 from D(1), S(1) to D(2) & S(2)

Floyd–Warshall algorithm

How does it work?



k = 2

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5	1	4
3	2	5	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(2)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	5	1	4
3	2	5	-	2	9
4	4	1	2	-	3
5	7	4	9	3	-

We fill D(2) and S(2) with same rules as filling D(1) & S(1).

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	1	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(2)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	1	4	5
3	1	2	-	4	2
4	2	2	3	-	5
5	2	2	2	4	-

Floyd–Warshall algorithm

How does it work?

k = 3

D(1)	1	2	3	4	5
1	-	3	2	∞	∞
2	3	-	5	1	4
3	2	5	-	2	∞
4	∞	1	2	-	3
5	∞	4	∞	3	-

D(2)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	5	1	4
3	2	5	-	2	9
4	4	1	2	-	3
5	7	4	9	3	-

D(3)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	5	1	4
3	2	5	-	2	9
4	4	1	2	-	3
5	7	4	9	3	-

S(1)	1	2	3	4	5
1	-	2	3	4	5
2	1	-	1	4	5
3	1	2	-	4	5
4	1	2	3	-	5
5	1	2	3	4	-

S(2)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	1	4	5
3	1	2	-	4	2
4	2	2	3	-	5
5	2	2	2	4	-

S(3)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	1	4	5
3	1	2	-	4	2
4	2	2	3	-	5
5	2	2	2	4	-

Floyd–Warshall algorithm

How does it work?

k = 4

D(2)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	5	1	4
3	2	5	-	2	9
4	4	1	2	-	3
5	7	4	9	3	-

D(3)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	5	1	4
3	2	5	-	2	9
4	4	1	2	-	3
5	7	4	9	3	-

D(4)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-

S(2)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	1	4	5
3	1	2	-	4	2
4	2	2	3	-	5
5	2	2	2	4	-

S(3)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	1	4	5
3	1	2	-	4	2
4	2	2	3	-	5
5	2	2	2	4	-

S(4)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

Floyd–Warshall algorithm

How does it work?

k = 5

D(3)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	5	1	4
3	2	5	-	2	9
4	4	1	2	-	3
5	7	4	9	3	-

D(4)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-

S(3)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	1	4	5
3	1	2	-	4	2
4	2	2	3	-	5
5	2	2	2	4	-

S(4)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

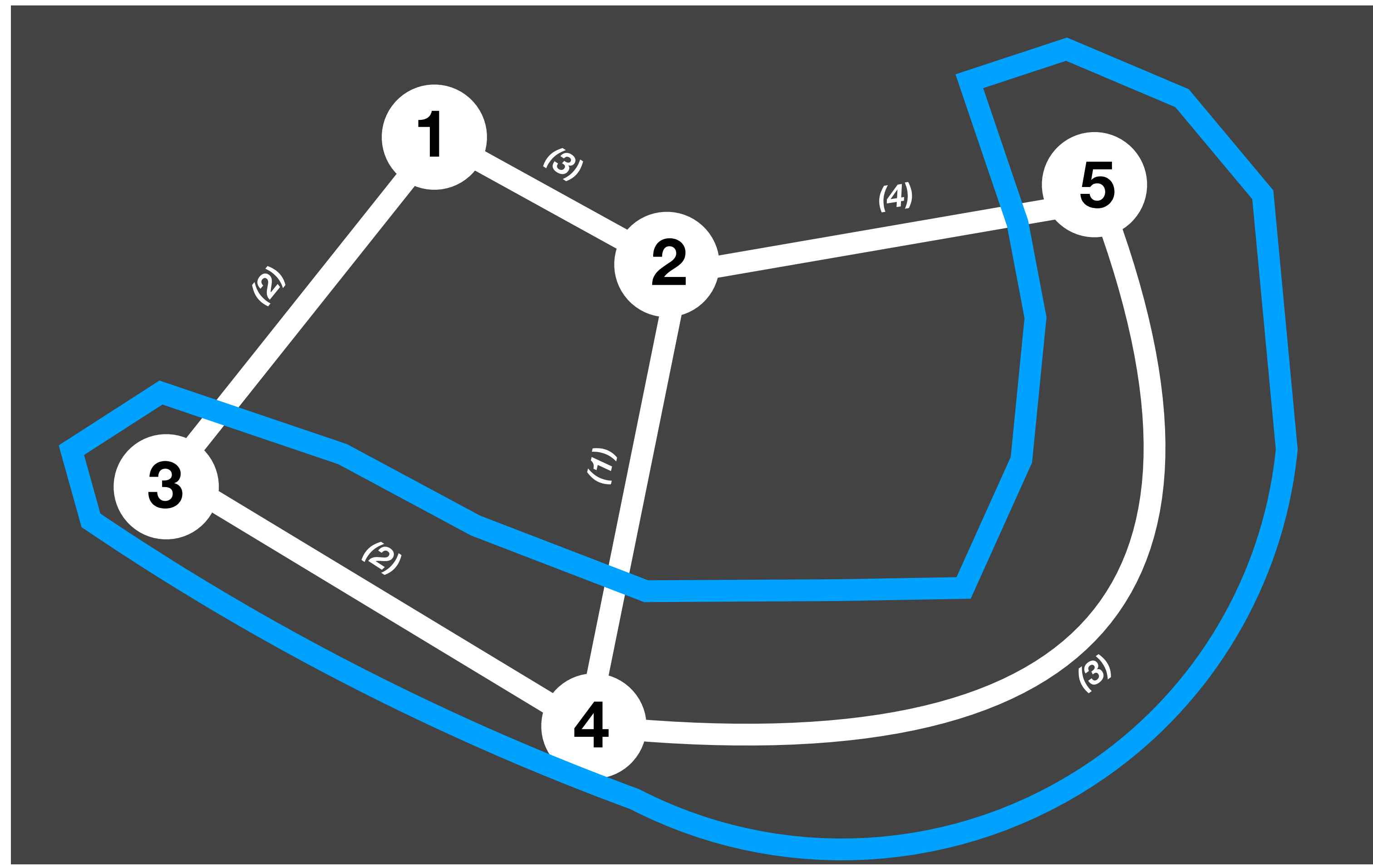
S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

Floyd–Warshall algorithm

How does it work?

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-

S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-



After k = max nodes, D(k) is shown as the shortest distance between any 2 nodes.

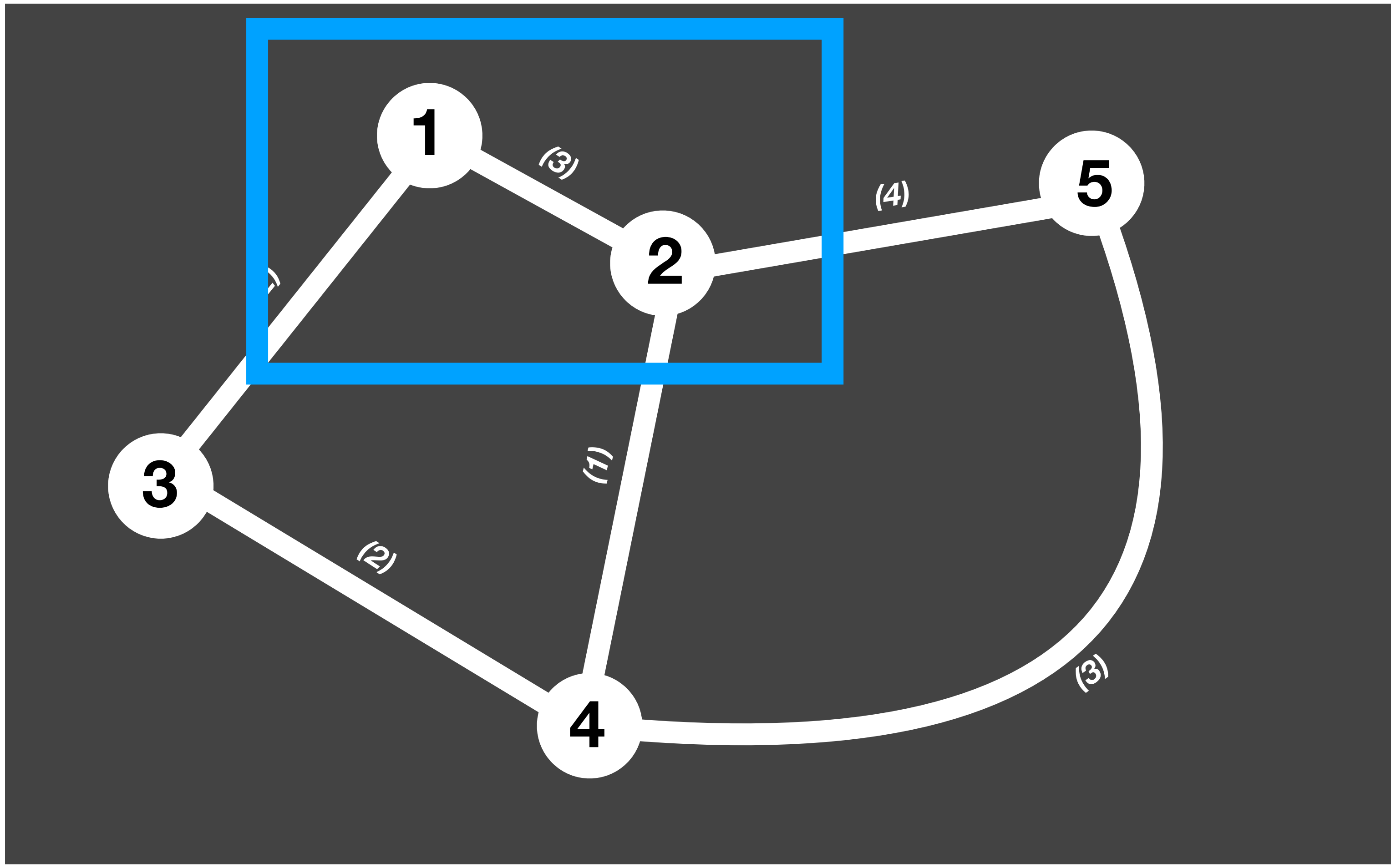
For example:
Shortest distance between node 3 and node 5 is **5.**
Shortest distance between node 2 and node 1 is **3.**

Floyd–Warshall algorithm

How does it work?

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-

S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-



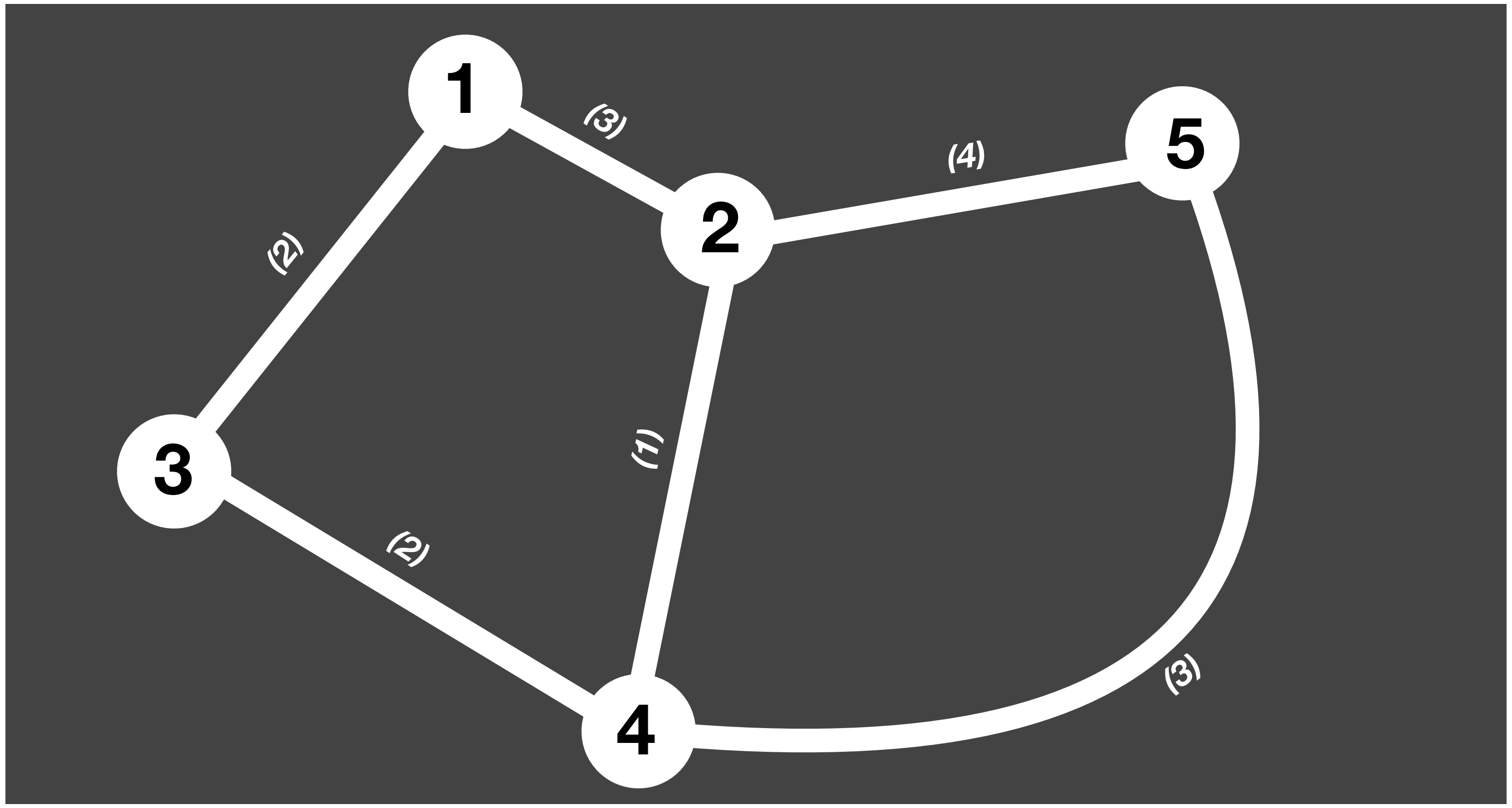
After k = max nodes, D(k) is shown as the shortest distance between any 2 nodes.

For example:
Shortest distance between node 3 and node 5 is **5**.
Shortest distance between node 2 and node 1 is **3**.

Floyd–Warshall algorithm

How does it work?

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-



S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

Next, to check the part from 1 node to another node, we using this method:
For example:

Shortest part from 1 to 2 is 3

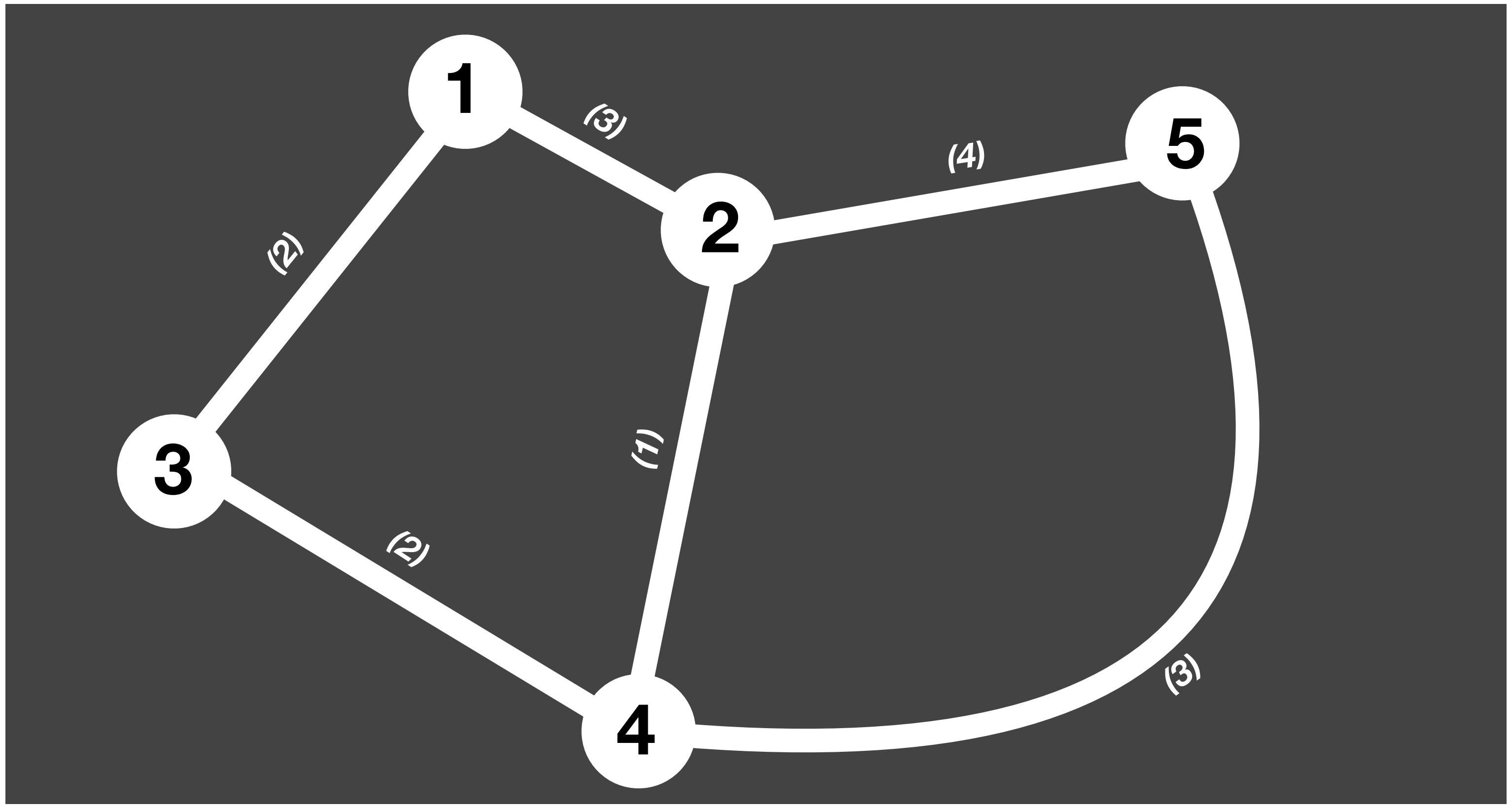
1 -> 2

We are looking into row 1 column 2 of S(5).
If the value is equal to its column, then there will be direct way from 1 to 2

Floyd–Warshall algorithm

How does it work?

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-



S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

Shortest part from 1 to 4 is 4

But how can we track the way from node 1 to node 4?

1 -> 4

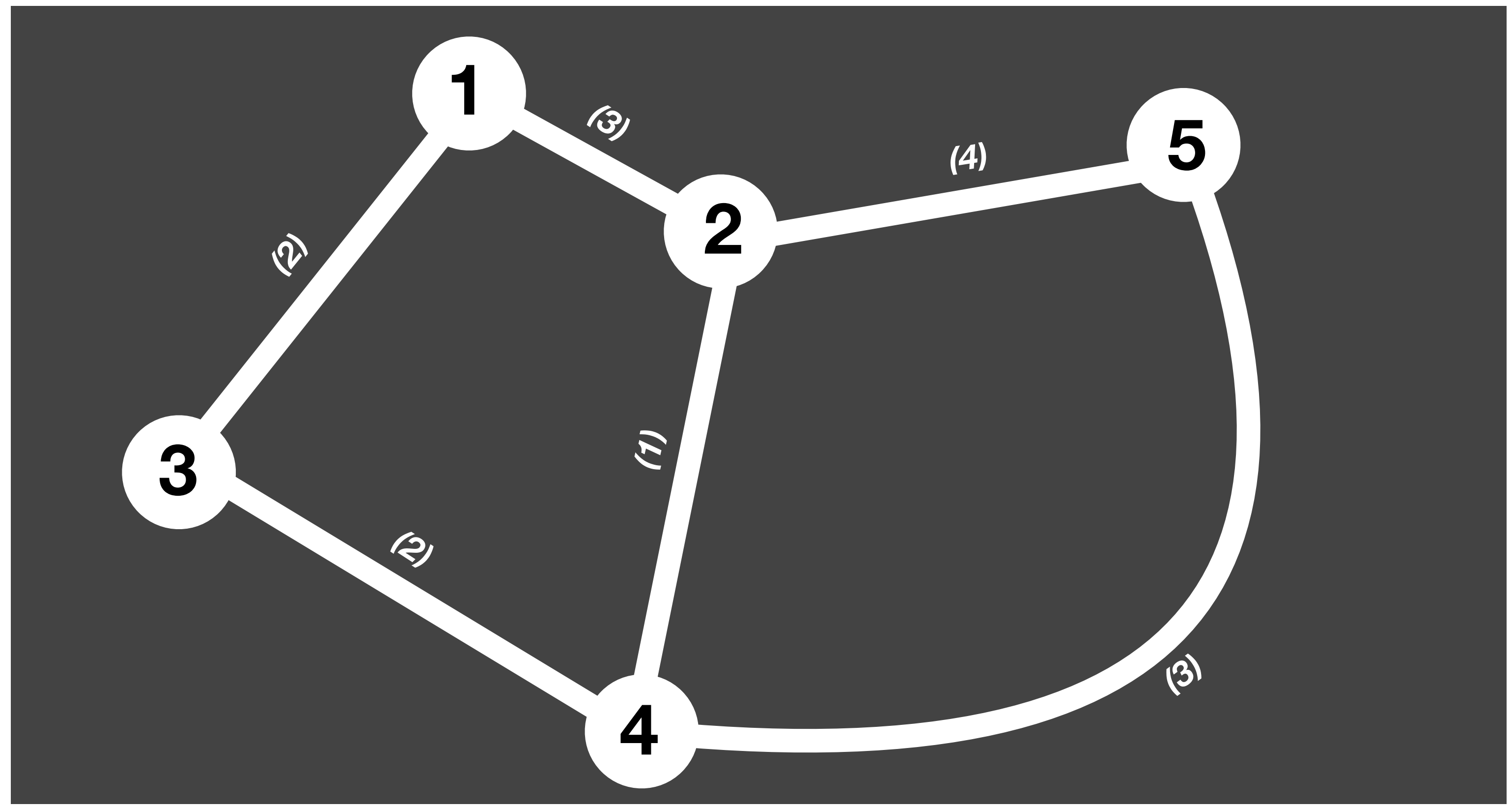
We are looking into row 1 column 4 of S(5).

The value is not the same with its column which means if you go from 1 to 4 you have to go through 2 first!

Floyd–Warshall algorithm

How does it work?

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-



S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

Shortest part from 1 to 4 is 4

We insert 2 into the track ray below:

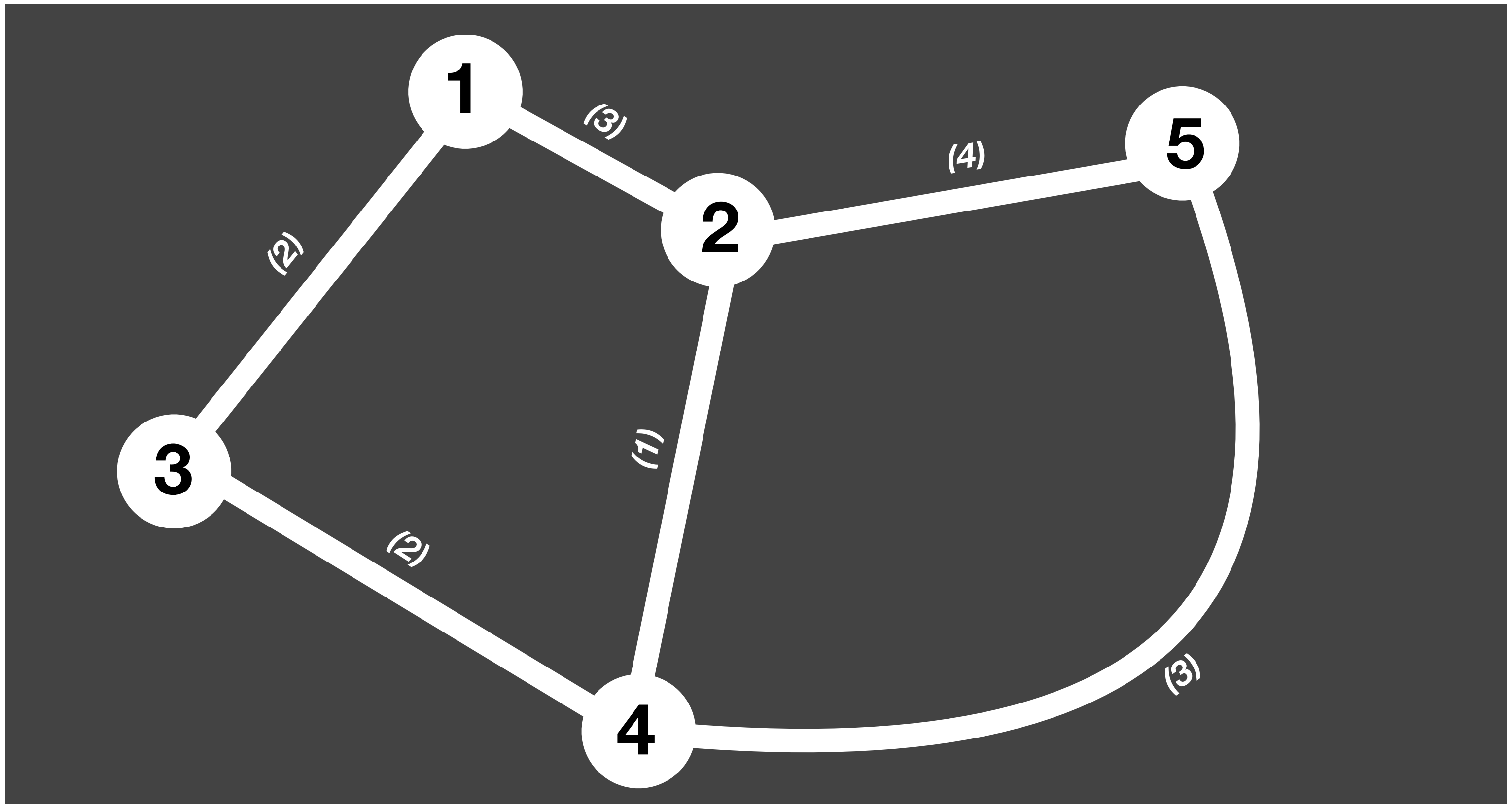
1 -> 2 -> 4

Then, we continue tracking the way from 2 to 4 and do the same until we find a value that match its column.

Floyd–Warshall algorithm

How does it work?

D(5)	1	2	3	4	5
1	-	3	2	4	7
2	3	-	3	1	4
3	2	3	-	2	5
4	4	1	2	-	3
5	7	4	5	3	-



S(5)	1	2	3	4	5
1	-	2	3	2	2
2	1	-	4	4	5
3	1	4	-	4	4
4	2	2	3	-	5
5	2	2	4	4	-

Shortest part from 1 to 4 is 4

1 -> 2 -> 4

Floyd–Warshall algorithm

Implementation

Build a program using Floyd algorithm to find shortest part from any 2 nodes.

Request input:

- First line contains n and m. With n is number of nodes and m is number of edges.
- Next m-th lines contain 3 numbers: a and b and c. With a & b is the edge connects between node a and b with weight c.

Request output:

- Shortest part from every 2 nodes and distinguish.

Example input:

```
5 6
1 2 3
1 3 2
2 5 4
2 4 1
3 4 2
4 5 3
```

Example output:

```
1 2 3
1 3 2
1 4 4
1 5 7
2 3 3
2 4 1
2 5 4
3 4 2
3 5 5
4 5 3
```

Floyd–Warshall algorithm

Implementation

```
void input()
{
    cin >> n >> m; // Get number of nodes and edges
    for (int i = 1; i<=n; i++)
    for (int j = 1; j<=n; j++)
      a[i][j] = MAX_INT;  //No distance, marked as
      infinity distance

    for (int i = 1; i<=m; i++)
    {
      cin >> numa >> numb >> c;
      if (c <= a[numa][numb]) a[numa][numb] = c;
      // Remove parallel edge(s)
      a[numb][numa] = a[numa][numb];
    };

    for (int i = 1; i<=n; i++) a[i][i] = -1;
    //Remove loops
};
```

Floyd–Warshall algorithm

Implementation

```
void floyd()
{
    for (int k = 1; k<=n; k++)
    for (int i = 1; i<=n; i++)
    for (int j = 1; j<=n; j++)
     if (a[i][j] > a[i][k] + a[k][j]) then
      a[i][j] = a[i][k] + a[k][j];

};
```

Floyd won't work with a graph that has:

A. Positive edges
B. Negative edges
C. Positive cycles
D. Negative cycles

Floyd won't work with a graph that has:

A. Positive edges
B. Negative edges
C. Positive cycles
D. Negative cycles

Time complexity of Floyd?

A. $O(n)$
B. $O(n \log(n))$
C. $O(n^2)$
D. $O(n^3)$

Time complexity of Floyd?

A. $O(n)$
B. $O(n \log(n))$
C. $O(n^2)$
D. **$O(n^3)$**

Time complexity of Floyd compare with Dijkstra algorithm finding shortest path?

A. Longer
B. Shorter

Time complexity of Floyd compare with Dijkstra algorithm finding shortest path?

A. Longer

B. Shorter

Conclusion

Using Floyd-Warshall algorithm to find shortest part from *any 2 nodes of the graph*.

Just need call the function **1 single time**. You will get all value and don't need call it for a second time.

Because this program contains 3 loops and time complexity will be $O(n^3)$, it will take more time either when number of nodes or edges is big (about > 1000 for nodes and > 100000 for edge).

Easy to implement code and easy to remember.

- End -