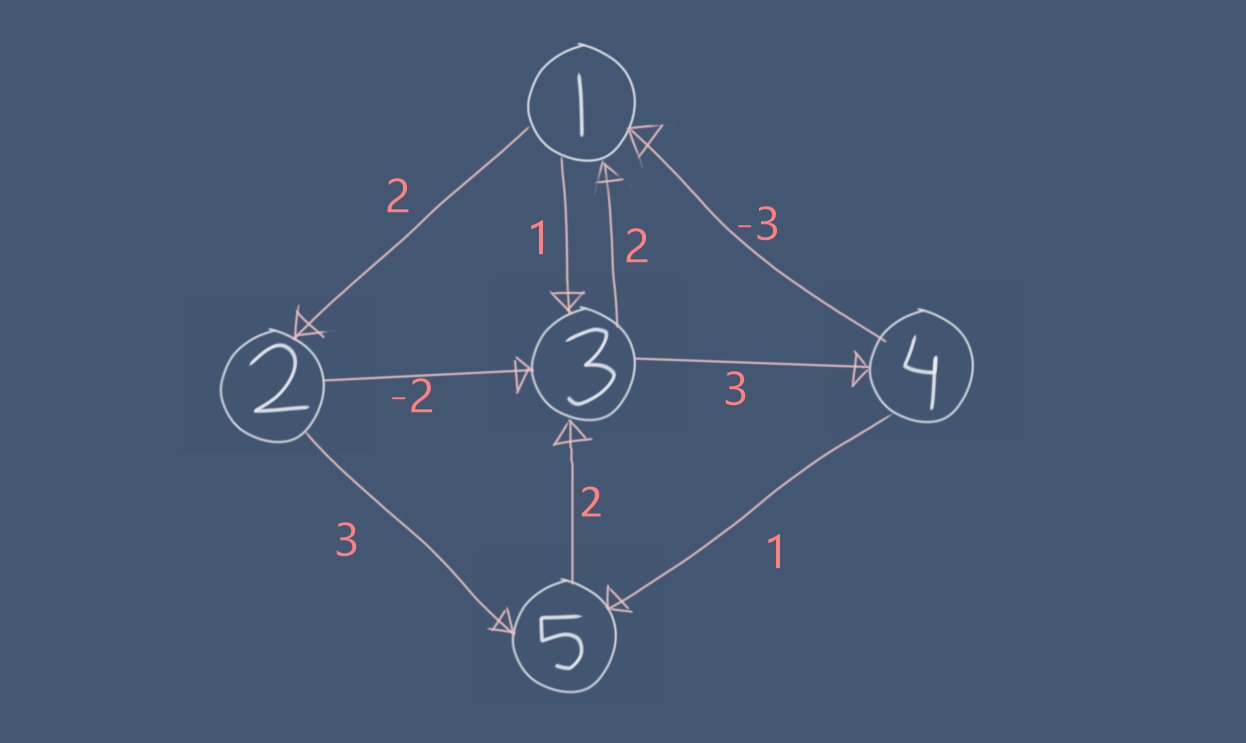
**Shortest Path**

The "shortest path" is the path from the start point to the end point with the smallest weight. When there is no path between the starting point and the end point, there is no path available and so there can not be any shortest path, there is no shortest path. The "shortest path" is not necessarily the path with the fewest points/nodes or the least points/nodes.

**BellMan Ford**

Bellman Ford is a **single source shortest path** graph algorithm. It is used to find the shortest paths from the source vertex to all other vertices in a weighted graph. It checks out for all possible solutions to pick up the best solution (uses **Brute force approach** & **Dynamic Programming technique**) and so it is slower but it can work with graphs in which edges can have negative weights.

Let’s see how it works first. Given a **weighted directed graph** below,

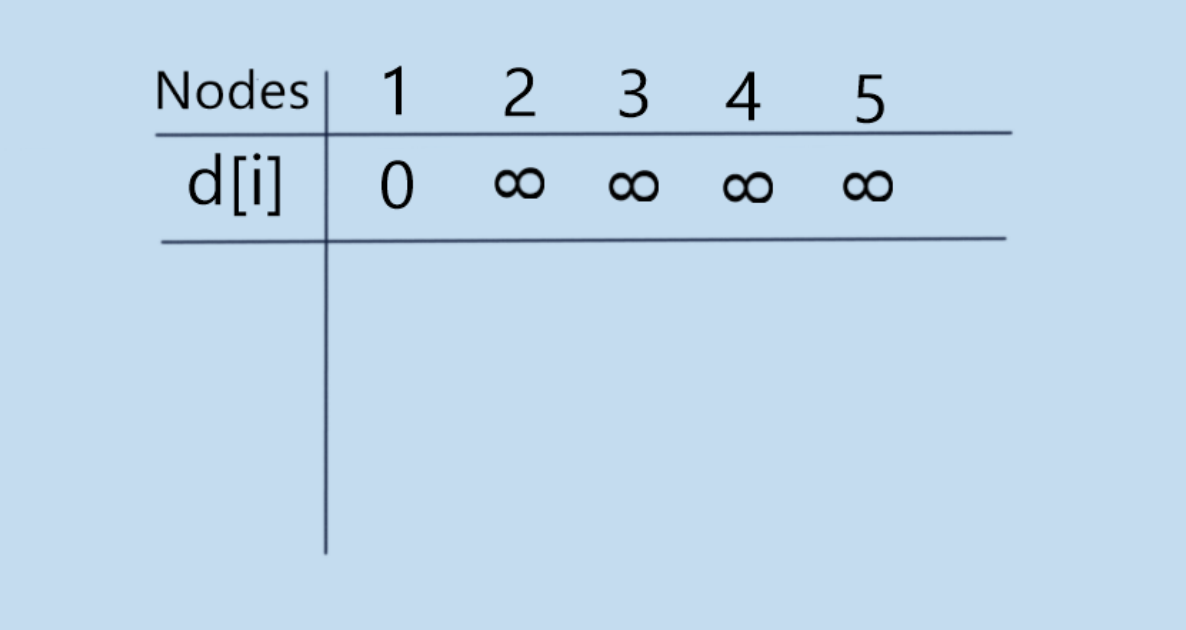


We have to relax all the edges of V-1 times (when there are V vertices or nodes).

We will make an edge list for that. For this example, the edgelist would be,

(1,2), (1,3), (2,3), (2,5), (3,1), (3,4), (4,1), (4,5), (5,3)

Now we can pick any node from the graph which would be the source (Remember we said it is a single source shortest path algorithm). Here I’m picking 1 as the source. Then we assume other nodes distance from the source are Infinity (∞).



After that we pick edges one by one from our edgelist we have created a few moments ago which are like (u,v) and then we start using our Relax method which can be represented in a pseudo code like this,

Relax (u, v, w) {

If ((d[u] + w) < d[v])

d[v] = d[u] + w

}

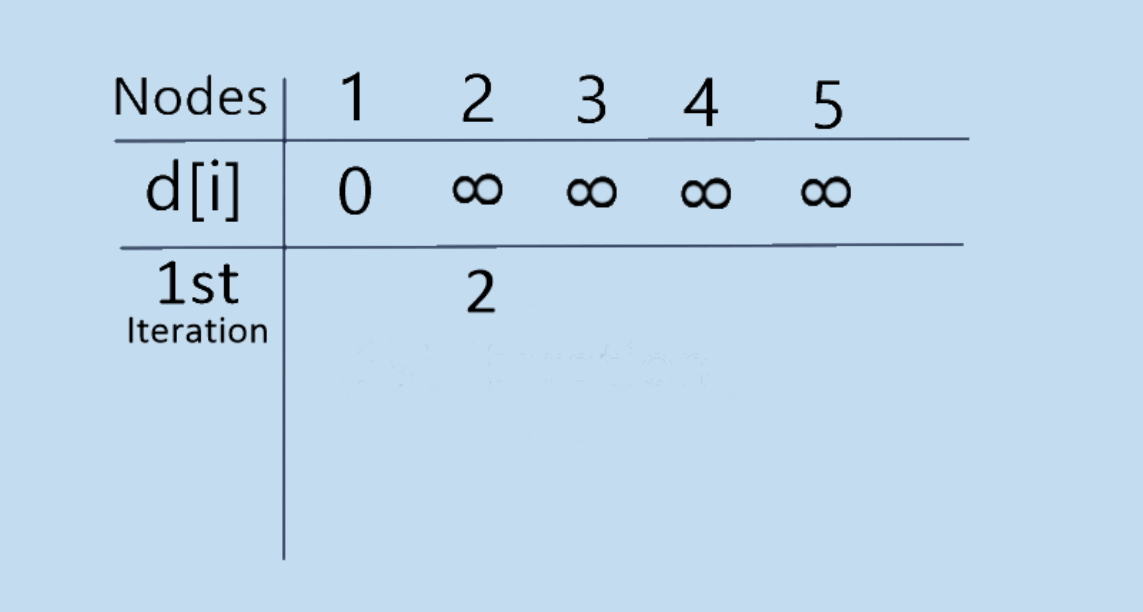
Here,

u & v means the nodes that edge is connected to

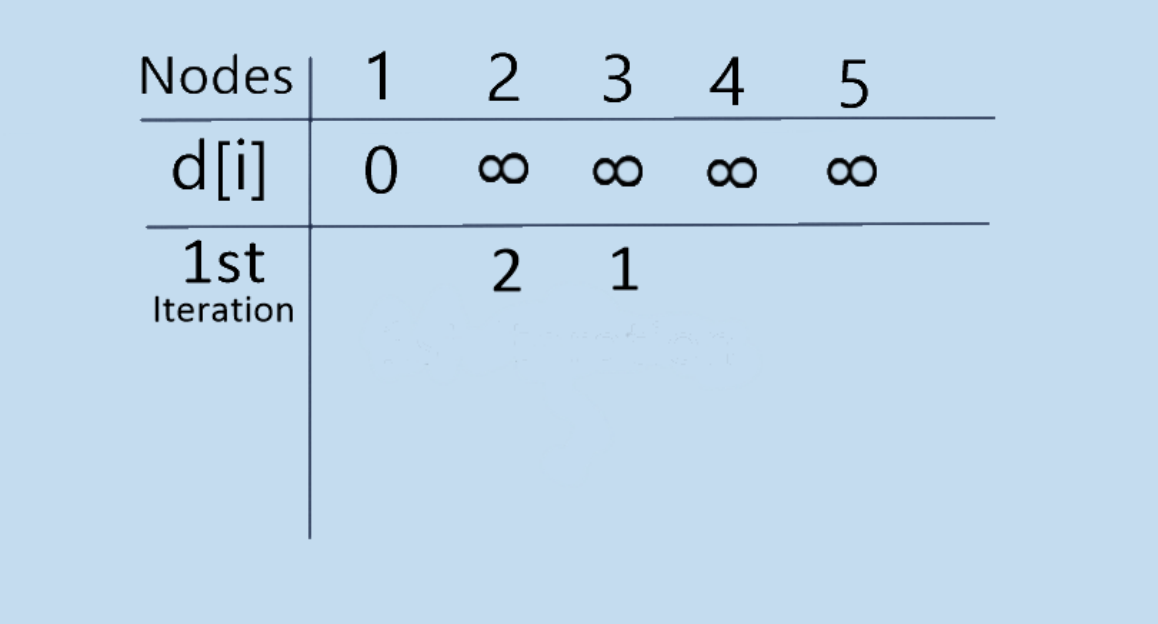
w means the weight of that edge

Assume that we are picking (1,2) first so the values of the variables would be

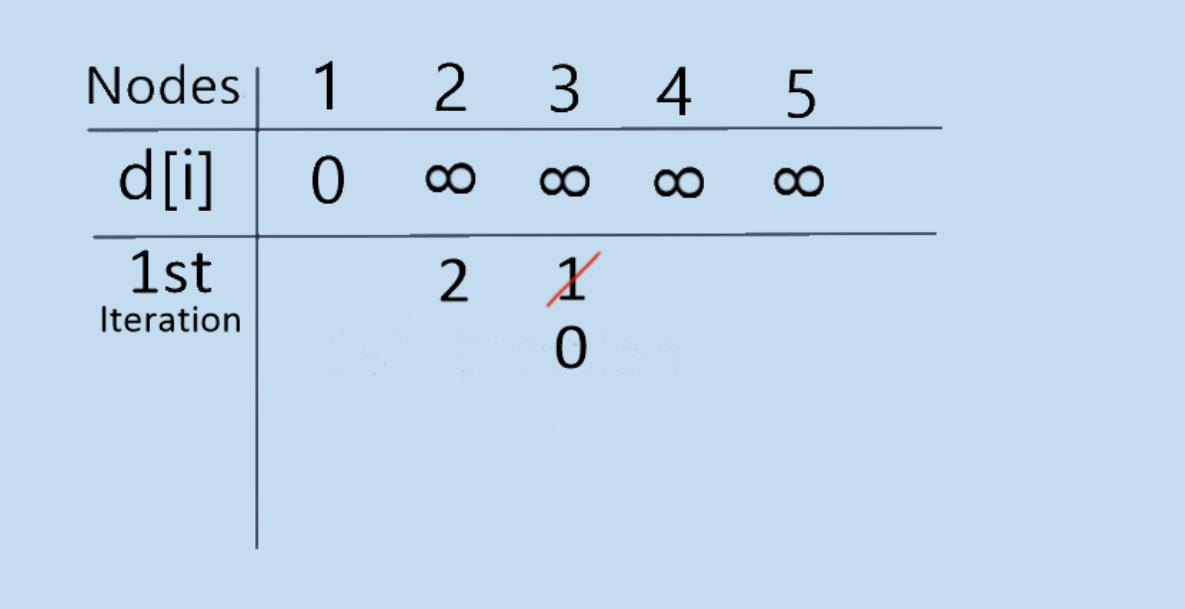
u = 1, v = 2, w = 2. Here, d[u] = 0 (look at the table above) and so d[u] + w = 2. It’s definitely greater than d[v] (as d[v] = ∞). So, we update the value of d[v].



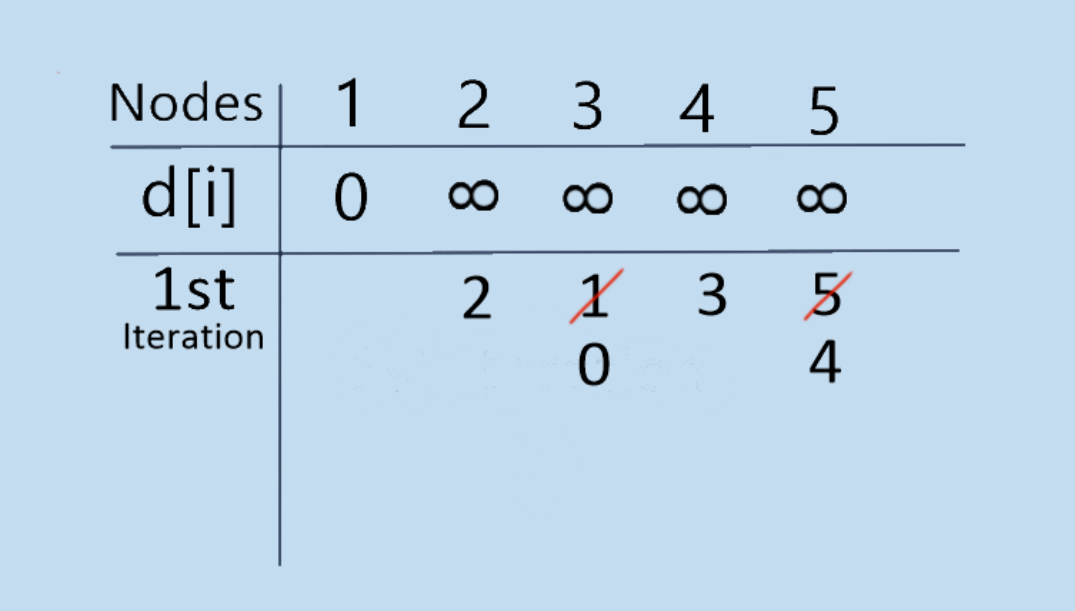
The next (1,3). As d[u] + w = 1 less than ∞, we update.



Then it comes to (2,3). Here d[u] = d[2] = 2 as we have updated node 2 recently. w = -2 (look at the graph). So, d[u] + w = 0. d[v] = d[3] = 1 and 0 is definitely less than 1. So, we update the table.

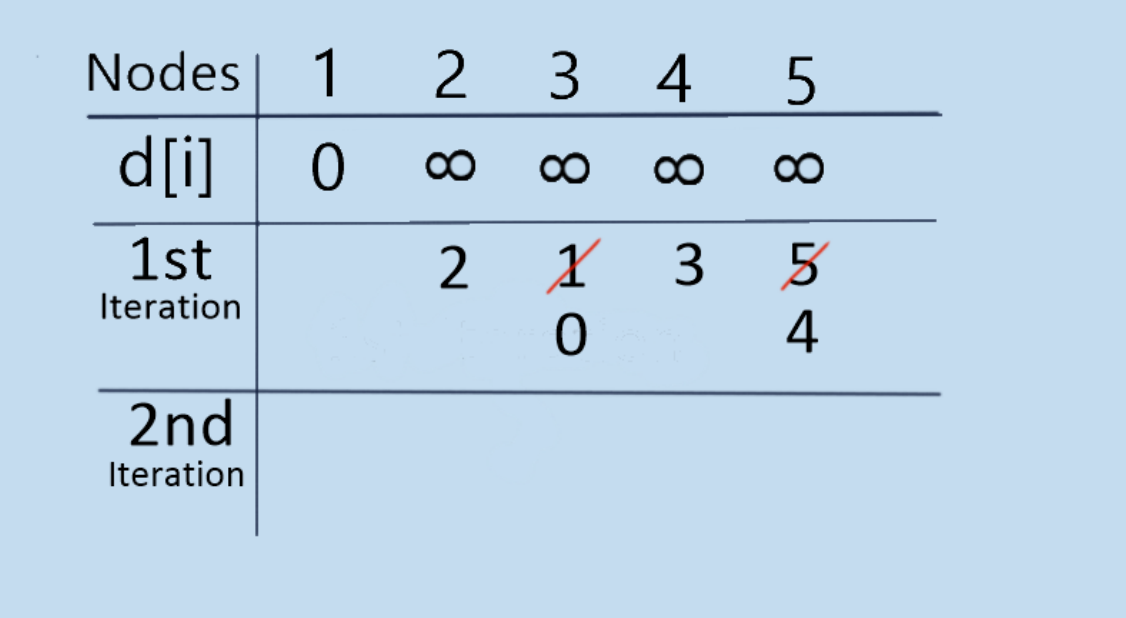


Similarly, we keep doing the same thing for all other edges and then our table looks like this. (Remember we don’t update the table when the edge can’t fulfill its condition in Relax method/function)



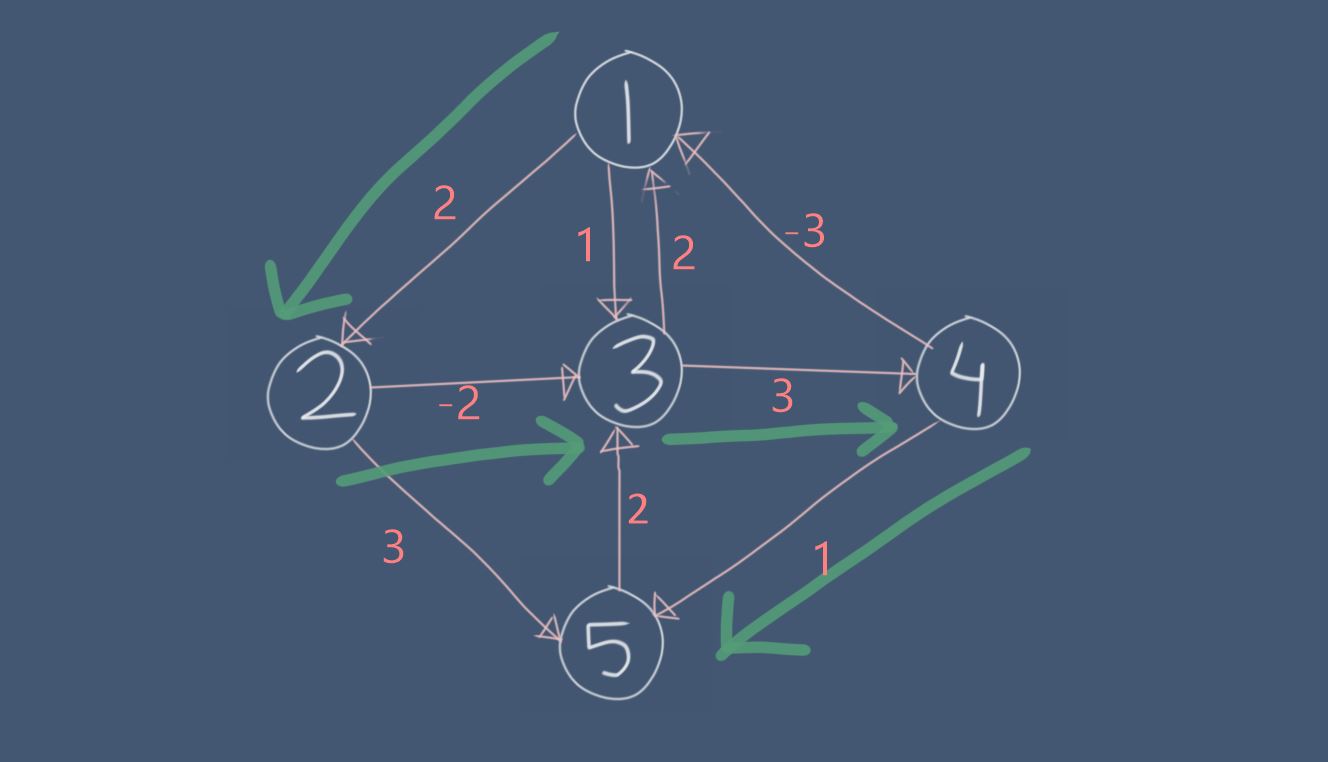
Thus, our first iteration is finished.

Now we have to start our second iteration and we have to keep doing the same thing for V-1 times (here V = 5 so, 4 times for this graph).



But after checking all the the edges we can’t find any changes at all. So, for this graph, we break after the 2nd iteration.

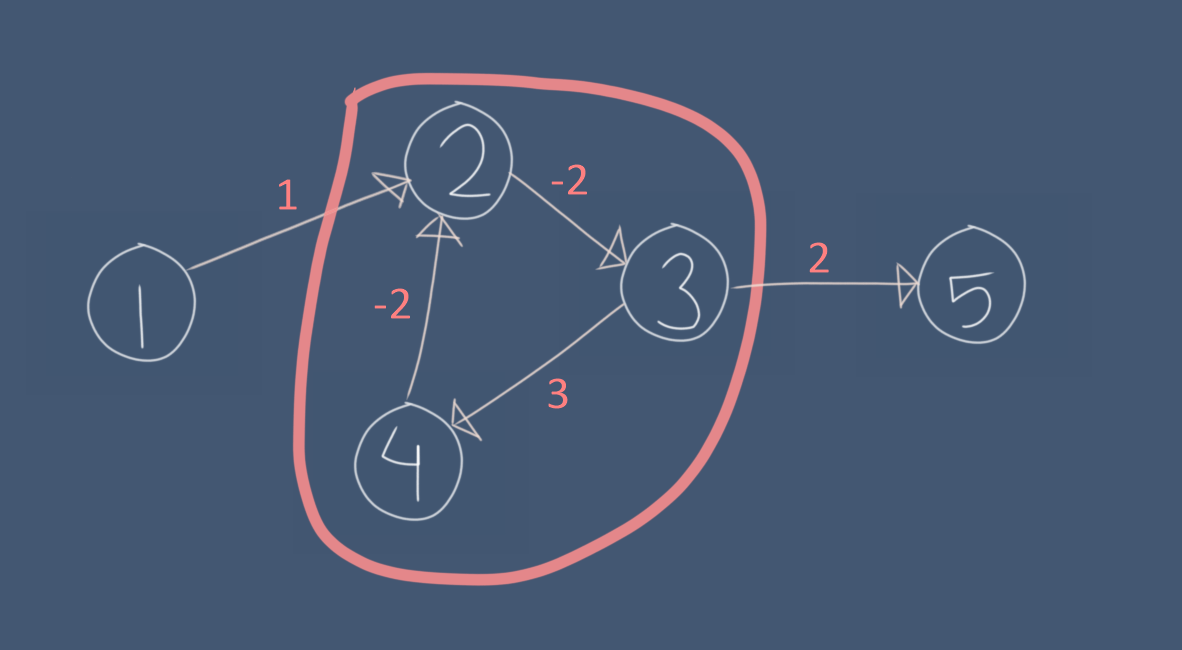
Now we have got distances of shortest paths to all the nodes/vertices from a single source (node 1). For example, the shortest path from 1 to 5 would be of 4 weights.



**Negative cycles:**

A negative cycle is one in which the overall sum of the cycle comes negative.

For example,



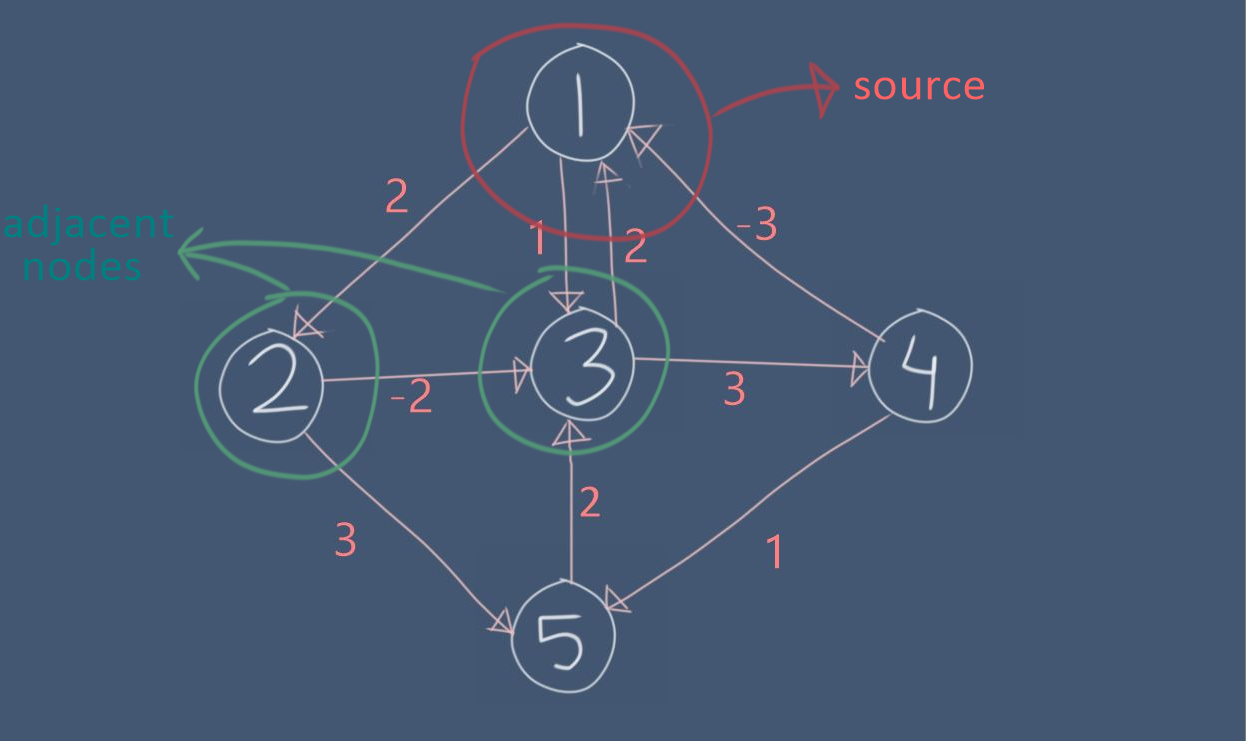
This graph has a cycle and the sum of the edges of the cycle is (-2) + (-2) + 3 = -1. So, if we want to find a shortest path from node 1 to node 5, we can just keep going 2->3->4->2->3… because this cycle keeps making the net weight of the path smaller and smaller. Hence, there is no shortest path available in graphs with negative edges.

The advantage of using Bellman Ford is that it can detect whether a graph has negative edges or not. We know that, there shouldn’t be any change in distances of the nodes from the source after (V-1) times iteration. But sometimes it still fulfills the if condition and changes the value of d[v]. When that happens, we can be sure that the graph has at least one negative cycle and finding shortest path is not possible for this graph.

Negative weights might seem absurd at first but they can explain a lot of phenomena like cashflow, heat released/absorbed in a chemical reaction etc. For instance, if there are different methods/reactions to make Chemical B from A, then each method will have sub-reactions involving both heat releasing and absorption. Bellman Ford is used in Optimization in Chemical Production.

**Noteworthy points**

* After the 1st iteration we become sure about the shortest path of the adjacent nodes to the source.



Then after the 2nd iteration we become sure of those nodes which are adjacent to the nodes that were adjacent to the source. And so on…

* If there are V nodes and more than V-1 edges then there must be a cycle in that graph. We don’t need to calculate cycles to calculate shortest path. So, we know for sure that there would be at most V-1 iterations

**Pseudo Code:**

BellmanFord (G,w,s)

for (x: x ∈ V)

d[x] = INF

d[s] = 0

for (i from 1 to V-1)

for ((u,v,w): (u,v,w) ∈ EdgeList)

Relax(u,v,w)

for ((u,v,w): (u,v,w) ∈ EdgeList)

if (d[u] + w < d[v])

print (“Negative cycle is present”)

return false

return true

Relax (u,v,w)

If (d[u] + w < d[v])

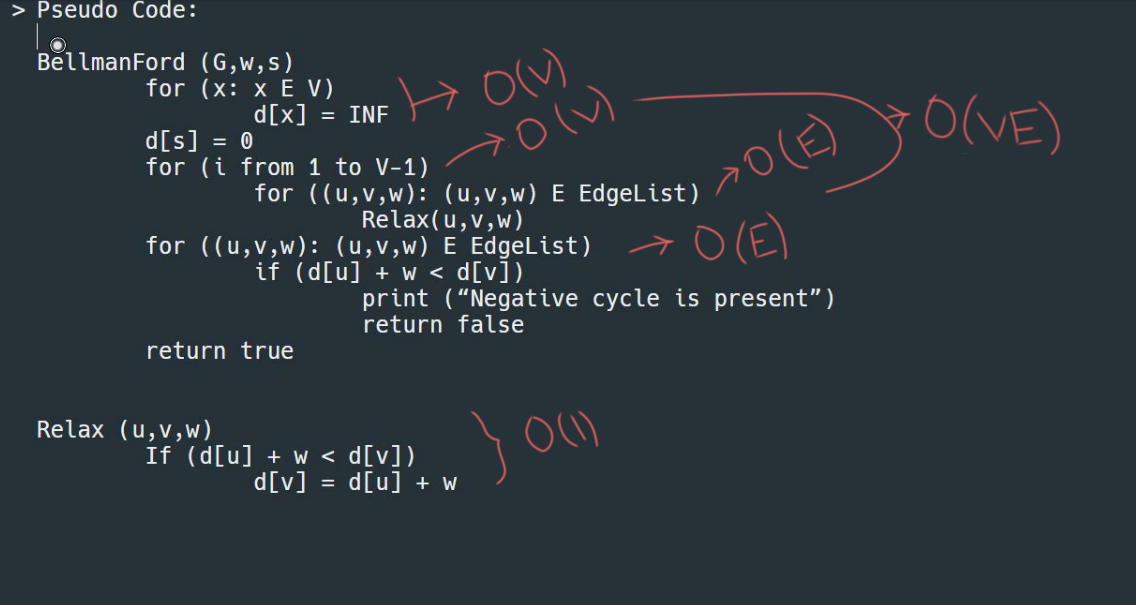
d[v] = d[u] + w

**Time Complexity:**

Let,

V = number of Vertices/Nodes

E = number of Edges



So, the time complexity would be O(V + VE+ E) = **O (VE)**

**Memory Complexity =** O(V+E)