Shortest Path Algorithms

Two types:

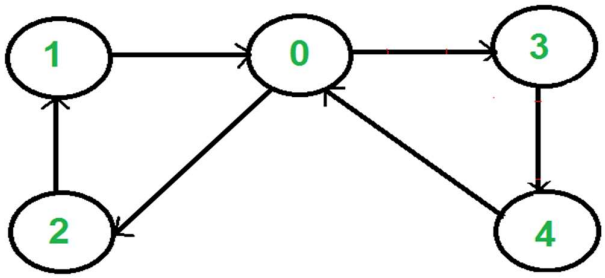
1. Single source shortest path algorithm – Finds shortest path from single source vertex to all other vertices in the graph.
   Example: Dijkstra's algorithm, Bellman ford algorithm
2. All-pair shortest path algorithm – Finds shortest path between all pairs of vertices.
   Example: Floyd Warshall Algorithm

Bellman Ford Algorithm:

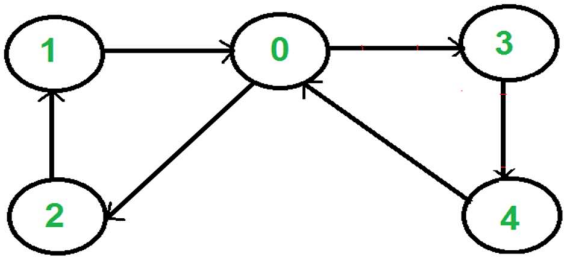
- Bellman ford algorithm is a single-source shortest path algorithm. Computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.
- It is capable of handling graphs in which some of the edge weights are negative numbers.
- If the weighted graph contains the negative weight values, then the Dijkstra's algorithm does not confirm whether it produces the correct answer or not.
- Bellman ford algorithm guarantees the correct answer even if the weighted graph contains the negative weight values.
- Weight of edges can represent everything in real world. Example: Consider a graph simulating behaviour of a molecule in a chemical reaction (i.e.) which paths it can take during reaction. Weights represents energy absorbed or released in the transition. We represent released energy with +ve weights and absorbed energy with -ve.
- Negative cycles can also be detected using Bellman Ford algorithm.

Basic terms related to Bellman Ford algorithm and examples:

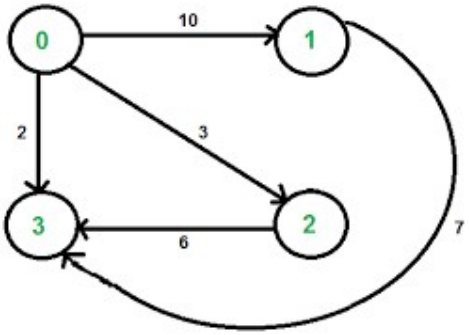
- Cycle – Closed loop



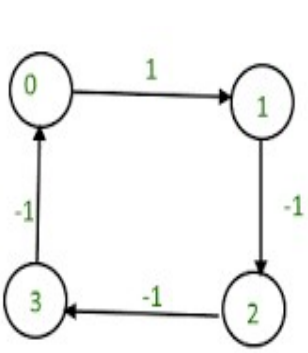
- Example Graph with cycles

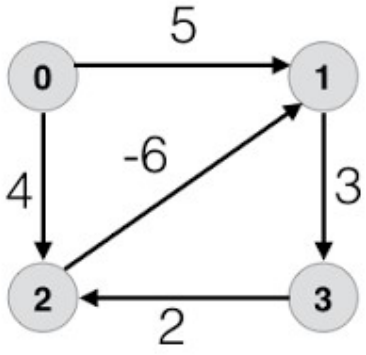


- Example Graph without cycles



- Negative cycle - Sum of costs of all the edges in the cycle is negative.





Idea behind Bellman Ford algorithm:

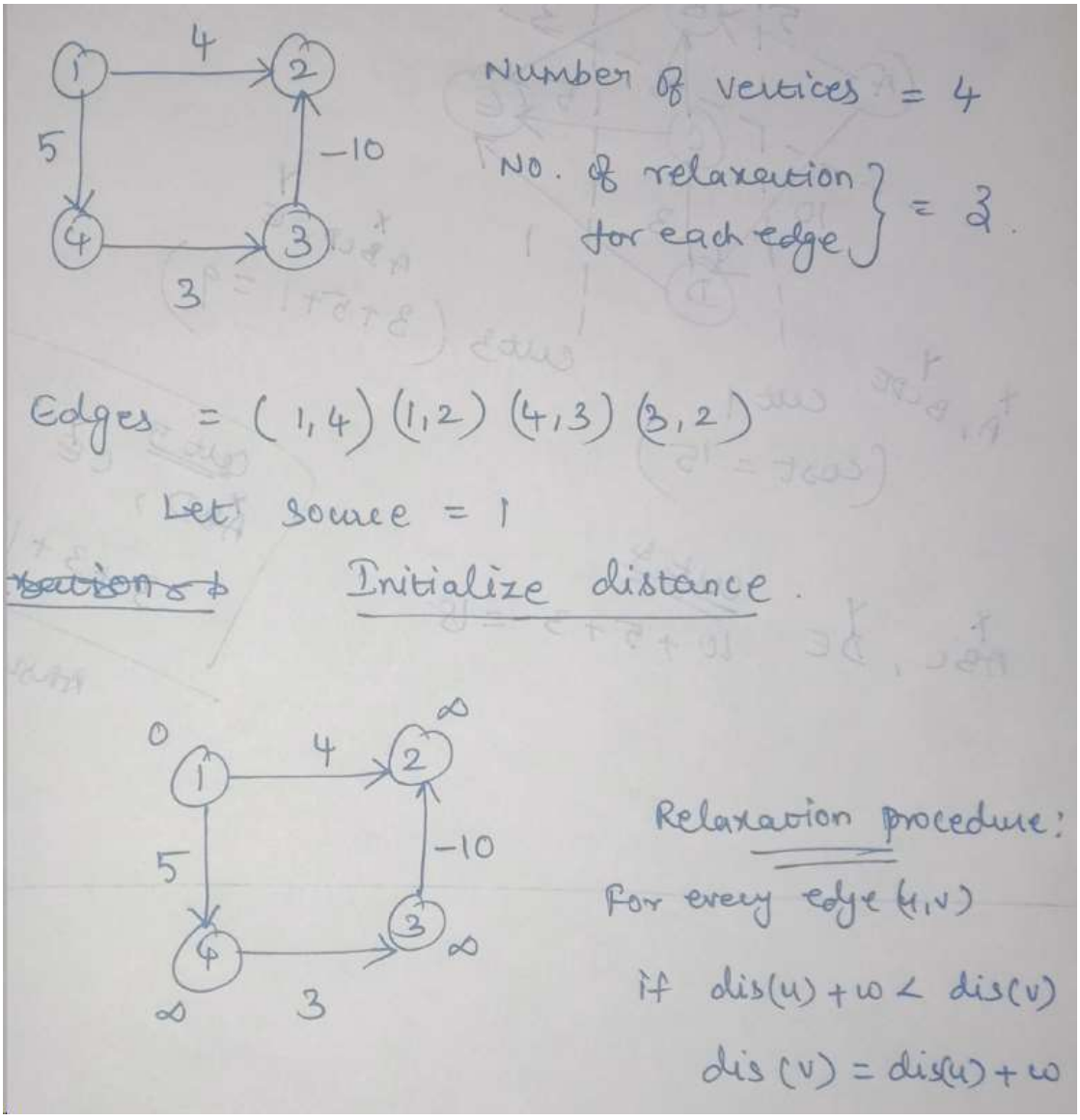
- Relax all the edges (n-1) times where, n = number of vertices in the given graph.
- **Relaxing procedure:**

  For every edge (u,v)

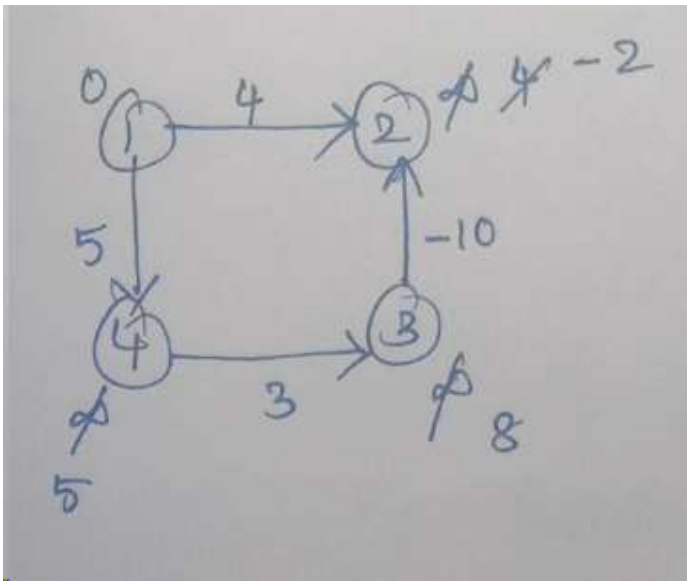
  if distance[u] + w < distance[v] **then**

  distance[v] := distance[u] + w

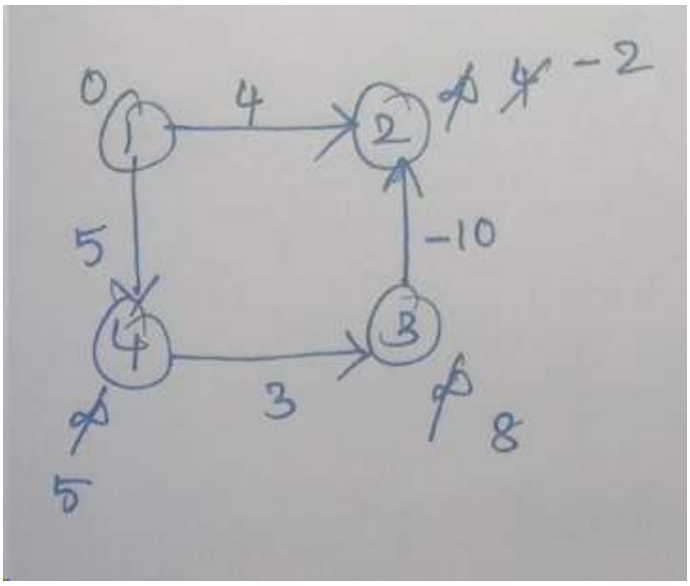
Example:



Relaxation-1:



Relaxation-2:



As a result of relaxation-2, no changes in the distance occurred. At this point, we can stop relaxing (i.e.) need not do relaxation-3.

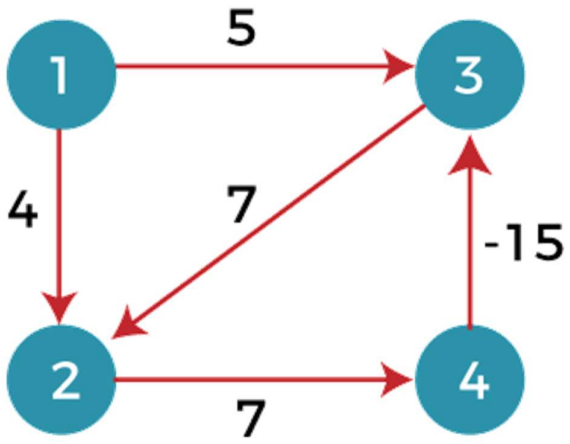
Final result:

Distance[1] = 0, Distance[2] = -2, Distance[3] = 8, Distance[4] = 5

Drawback of Bellman ford algorithm:

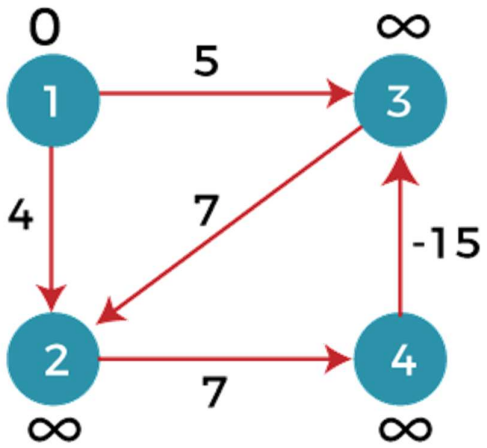
- The bellman ford algorithm does not produce a correct answer if the graph contains negative cycle (i.e.) sum of the edges of a cycle is negative.

Consider the below graph.



we consider vertex 1 as the source vertex.

Initialize distance as shown below:

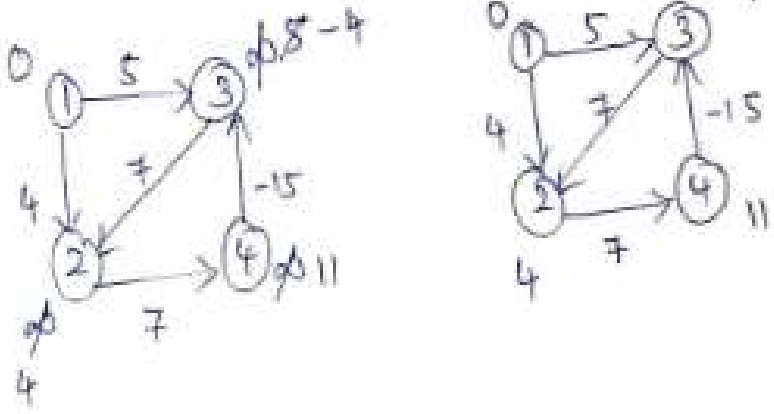


Edges: (1, 2), (1, 3), (2, 4), (3, 2), (4, 3)

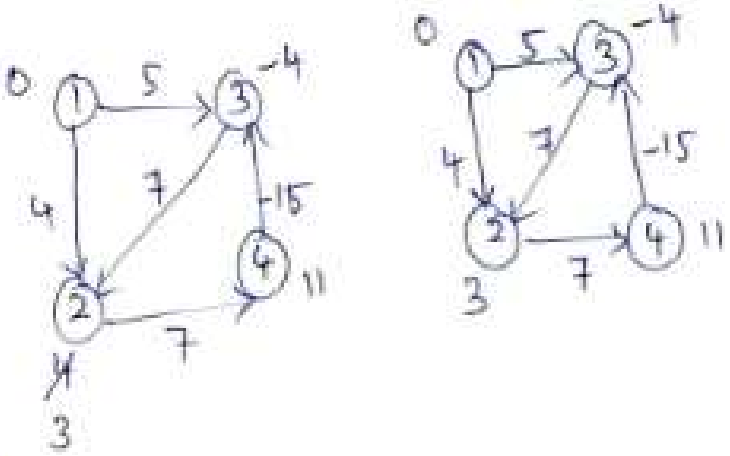
Number of vertices = 4

Number of relaxations = 3

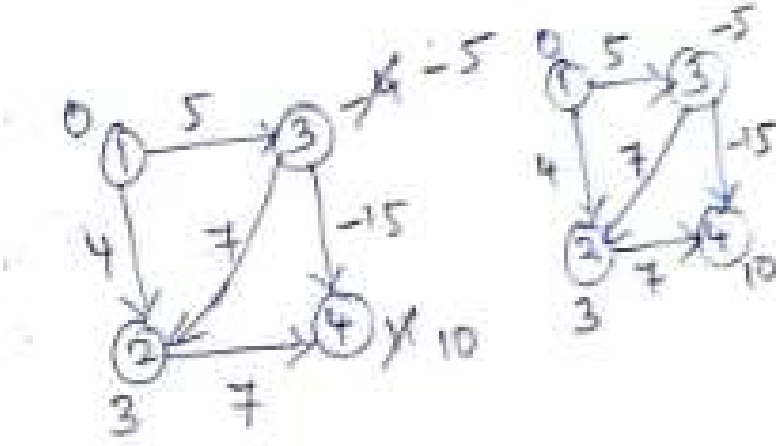
Relaxation-1:



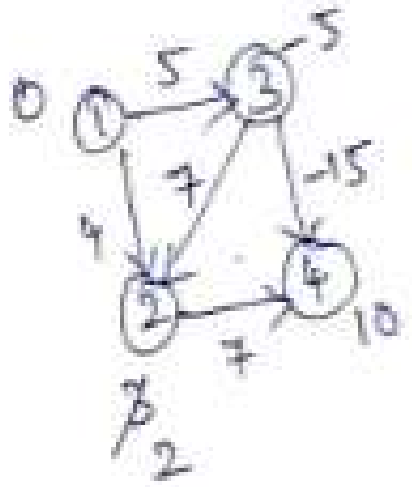
Relaxation-2:



Relaxation-3:



Relaxation-4: To check for negative cycles



Final result:

Graph contains a negative-weight cycle. So, no solution.

Important facts:

- Relaxes each edge number of vertices – 1 times
- Relaxing procedure:

  For every edge (u,v)

  if distance[u] + w < distance[v] **then**

  distance[v] := distance[u] + w
- Once, relaxation is completed, detects negative cycles by checking following 'if' condition with respect to each edge. 'if' condition is nothing but the same relaxing condition.

  if distance[u] + w < distance[v] **then**

  Print "Graph contains a negative-weight cycle. So, no solution"

Algorithm:

function Bellman_Ford(*list* vertices, *list* edges, *vertex* source)

// Step 1: initialize graph

for each vertex v **in** vertices **do**

distance[v] := **inf** // Initialize the distance to all vertices to infinity

distance[source] := 0 // The distance from the source to itself is, of course, zero

// Step 2: relax edges repeatedly

for i **from** 1 **to** size(vertices)−1 **do** // |V|−1 repetitions (i.e) n-1 relaxations where n is the number of vertices.

for each edge (u, v) **with** weight w **in** edges **do**

if distance[u] + w < distance[v] **then**

distance[v] := distance[u] + w

//Step 3: to check if the graph contains any negative weight cycle, perform n^{th} relaxation

Set flag=0

for each edge (u, v) **with** weight w **in** edges **do**

if distance[u] + w < distance[v] **then**

Print “Graph contains negative weight cycle; so, no solution"

flag=1;

//Step 4: Final output

if(flag==0)

for i **from** 1 **to** size(vertices) **do**

Print distance[i]

Time complexity: O(VE) where V = number of vertices and E = number of edges.

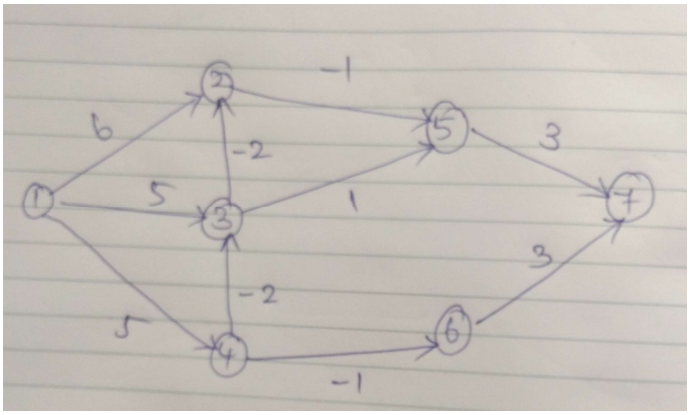
Note:

We can generate all pair shortest paths if we run the bellman algorithm from each node and get shortest paths to all other vertices.

Time complexity: O(VVE)

Practice Problem:

Find the shortest path from Vertex 1 using Bellman Ford Algorithm and show the result.



Solution: