Bellman–Ford Algorithm - GeeksforGeeks

Source: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Bellman–Ford Algorithm Last Updated : 23 Jul, 2025 Given a weighted graph with V vertices and E edges, along with a source vertex src , the task is to compute the shortest distances from the source to all other vertices. If a vertex is unreachable from the source, its distance should be marked as 10 8 . In the presence of a negative weight cycle, return -1 to signify that shortest path calculations are not feasible. Examples: Input : V = 5, edges = [[0, 1, 5], [1, 2, 1], [1, 3, 2], [2, 4, 1], [4, 3, -1]], src = 0 Output : [0, 5, 6, 6, 7] Explanation: Shortest Paths: For 0 to 1 minimum distance will be 5. By following path 0 $\rightarrow$ 1 For 0 to 2 minimum distance will be 6. By following path 0 $\rightarrow$ 1 $\rightarrow$ 2 For 0 to 3 minimum distance will be 6. By following path 0 $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 4 $\rightarrow$ 3 For 0 to 4 minimum distance will be 7. By following path 0 $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 4 Input : V = 4, edges = [[0, 1, 4], [1, 2, -6], [2, 3, 5], [3, 1, -2]], src = 0 Output : [-1] Explanation: The graph contains a negative weight cycle formed by the path 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 1 , where the total weight of the cycle is negative. Try it on GfG Practice Table of Content Bellman-Ford Algorithm - O(V*E) Time and O(V) Space Negative weight cycle: Limitation of Dijkstra's Algorithm: Principle of Relaxation of Edges Why Relaxing Edges (V - 1) times gives us Single Source Shortest Path? Detection of a Negative Weight Cycle Problems based on Shortest Path Approach: Bellman-Ford Algorithm - O(V*E) Time and O(V) Space Negative weight cycle: A negative weight cycle is a cycle in a graph, whose sum of edge weights is negative. If you traverse the cycle, the total weight accumulated would be less than zero. In the presence of negative weight cycle in the graph, the shortest path doesn't exist because with each traversal of the cycle shortest path keeps decreasing . Limitation of Dijkstra's Algorithm: Since, we need to find the single source shortest path , we might initially think of using Dijkstra's algorithm . However, Dijkstra is not suitable when the graph consists of negative edges . The reason is, it doesn't revisit those nodes which have already been marked as visited. If a shorter path exists through a longer route with negative edges, Dijkstra's algorithm will fail to handle it. Principle of Relaxation of Edges Relaxation means updating the shortest distance to a node if a shorter path is found through another node. For an edge (u, v) with weight w : If going through u gives a shorter path to v from the source node (i.e., distance[v] > distance[u] + w), we update the distance[v] as distance[u] + w. In the bellman-ford algorithm, this process is repeated (V - 1) times for all the edges. Why Relaxing Edges (V - 1) times gives us Single Source Shortest Path? A shortest path between two vertices can have at most (V - 1) edges. It is not possible to have a simple path with more than (V - 1) edges (otherwise it would form a cycle). Therefore, repeating the relaxation process (V - 1) times ensures that all possible paths between source and any other node have been covered. Assuming node 0 as the source vertex, let's see how we can relax the edges to find the shortest paths: In the first relaxation , since the shortest paths for vertices 1 and 2 are unknown (infinite, i.e., 10 8), the shortest paths for vertices 2 and 3 will also remain infinite (10 8) . And, for vertex 1, the distance will be updated to 4, as dist[0] + 4 < dist[1] (i.e., 0 + 4 < 10 8). In the second relaxation , the shortest path for vertex 2 is still infinite (e.g. 10 8), which means the shortest path for vertex 3 will also remain infinite. For vertex 2, the distance can be updated to 3, as dist[1] + (-1) = 3. In the third relaxation , the shortest path for vertex 3 will be updated to 5, as dist[2] + 2 = 5. So, in above example, dist[1] is updated in 1st relaxation , dist[2] is updated in second relaxation , so the dist for the last node (V - 1), will be updated in (V - 1) th relaxation . Detection of a Negative Weight Cycle As we have discussed earlier that, we need (V - 1) relaxations of all the edges to achieve single source shortest path. If one additional relaxation (Vth) for any edge is possible, it indicates that some edges with overall negative weight has been traversed once

more. This indicates the presence of a negative weight cycle in the graph. Bellman-Ford is a single source shortest path algorithm. It effectively works in the cases of negative edges and is able to detect negative cycles as well. It works on the principle of relaxation of the edges . Illustration: C++ #include <iostream> #include <vector> using namespace std ; vector < int > bellmanFord (int V , vector < vector < int >>& edges , int src) { // Initially distance from source to all // other vertices is not known(Infinite). vector < int > dist (V , 1e8); dist [src] = 0 ; // Relaxation of all the edges V times, not (V - 1) as we // need one additional relaxation to detect negative cycle for (int i = 0 ; i < V ; i ++) { for (vector < int > edge : edges) { int u = edge [0]; int v = edge [1]; int wt = edge [2]; if (dist [u] != 1e8 && dist [u] + wt < dist [v]) { // If this is the Vth relaxation, then there is // a negative cycle if (i == V - 1) return { -1 }; // Update shortest distance to node v dist [v] = dist [u] + wt ; } } } return dist ; } int main () { // Number of vertices in the graph int V = 5 ; // Edge list representation: {source, destination, weight} vector < vector < int >> edges = { { 1 , 3 , 2 }, { 4 , 3 , -1 }, { 2 , 4 , 1 }, { 1 , 2 , 1 }, { 0 , 1 , 5 } }; // Define the source vertex int src = 0 ; // Run Bellman-Ford algorithm to get shortest paths from src vector < int > ans = bellmanFord (V , edges , src); // Output the shortest distances from src to all vertices for (int dist : ans) cout << dist << " " ; return 0 ; } Java import java.util.Arrays ; class GfG { static int [] bellmanFord (int V , int [][] edges , int src) { // Initially distance from source to all other vertices // is not known(Infinite). int [] dist = new int [V] ; Arrays . fill (dist , (int) 1e8); dist [src] = 0 ; // Relaxation of all the edges V times, not (V - 1) as we // need one additional relaxation to detect negative cycle for (int i = 0 ; i < V ; i ++) { for (int [] edge : edges) { int u = edge [0] ; int v = edge [1] ; int wt = edge [2] ; if (dist [u] != 1e8 && dist [u] + wt < dist [v]) { // If this is the Vth relaxation, then there is // a negative cycle if (i == V - 1) return new int [] { - 1 }; // Update shortest distance to node v dist [v] = dist [u] + wt ; } } } return dist ; } public static void main (String [] args) { // Number of vertices in the graph int V = 5 ; // Edge list representation: {source, destination, weight} int [][] edges = new int [][] { { 1 , 3 , 2 }, { 4 , 3 , -1 }, { 2 , 4 , 1 }, { 1 , 2 , 1 }, { 0 , 1 , 5 } }; // Source vertex for Bellman-Ford algorithm int src = 0 ; // Run Bellman-Ford algorithm from the source vertex int [] ans = bellmanFord (V , edges , src); // Print shortest distances from the source to all vertices for (int dist : ans) System . out . print (dist + " "); } } Python def bellmanFord (V , edges , src): # Initially distance from source to all other vertices # is not known(Infinite) e.g. 1e8. dist = [100000000] * V dist [src] = 0 # Relaxation of all the edges V times, not (V - 1) as we # need one additional relaxation to detect negative cycle for i in range (V): for edge in edges : u , v , wt = edge if dist [u] != 100000000 and dist [u] + wt < dist [v]: # If this is the Vth relaxation, then there # is a negative cycle if i == V - 1 : return [- 1] # Update shortest distance to node v dist [v] = dist [u] + wt return dist if __name__ == '__main__' : V = 5 edges = [[1 , 3 , 2], [4 , 3 , - 1], [2 , 4 , 1], [1 , 2 , 1], [0 , 1 , 5]] src = 0 ans = bellmanFord (V , edges , src) print (' ' . join (map (str , ans))) C# using System ; class GfG { // Function to perform the Bellman-Ford algorithm static int [] bellmanFord (int V , int [,] edges , int src) { // Initialize distances from source to all // vertices as "infinity" (represented by 1e8) int [] dist = new int [V]; for (int i = 0 ; i < V ; i ++) dist [i] = (int) 1 e8 ; // Distance to the source is always 0 dist [src] = 0 ; // Get the number of edges from the 2D edge array int E = edges . GetLength (0); // Relax all edges V times // The extra iteration (V-th) is used to detect a // negative weight cycle for (int i = 0 ; i < V ; i ++) { for (int j = 0 ; j < E ; j ++) { // Extract edge info: from u to v with weight wt int u = edges [j , 0]; int v = edges [j , 1]; int wt = edges [j , 2]; // Only proceed if u has already been reached // (i.e., not infinity) if (dist [u] != (int) 1 e8 && dist [u] + wt < dist [v]) { // If this is the V-th iteration and relaxation is still // possible, it means there is a negative weight cycle if (i == V - 1) // Indicate presence of negative cycle return new int [] { - 1 }; // Update the distance to vertex v through vertex u dist [v] = dist [u] + wt ; } } } // Return the final shortest distances return dist ; } static void Main () { // Number of vertices in the graph int V = 5 ; // Edge list: each row represents {source, destination, weight} int [,] edges = { { 1 , 3 , 2 }, { 4 , 3 , - 1 }, { 2 , 4 , 1 }, { 1 , 2 , 1 }, { 0 , 1 , 5 } }; // Source vertex int src = 0 ; // Call Bellman-Ford and store the result int [] ans = bellmanFord (V , edges , src); // Print the shortest distances from source to all vertices foreach (int d in ans) Console . Write (d + " "); } } JavaScript function bellmanFord (V , edges , src) { // Initially distance from source to all // other vertices is not known(Infinite). let dist = new Array (V). fill (1e8); dist [src] = 0 ; // Relaxation of all the edges V times, not (V - 1) as we // need one additional relaxation to detect negative cycle for (let i = 0 ; i < V ; i ++) { for (let edge of edges) { let u = edge [0]; let v = edge [1]; let wt = edge [2]; if (dist [u] !== 1e8 && dist [u] + wt < dist [v]) { // If this is the Vth relaxation, then there is // a negative cycle if (i === V - 1) return [- 1]; // Update shortest distance to node v dist [v] = dist [u] + wt ; } } } return dist ; } // Driver Code let V = 5 ; let edges = [[1 , 3 , 2], [4 , 3 , - 1], [2 , 4 , 1], [1 , 2 , 1], [0 , 1 , 5]]; let src = 0 ; let ans = bellmanFord (V , edges , src); console . log (ans . join (" ")); Output 0 5 6 6 7 Problems based on Shortest Path Shortest Path in Directed Acyclic Graph Shortest path with one

curved edge in an undirected Graph Minimum Cost Path Path with smallest difference between consecutive cells Print negative weight cycle in a Directed Graph 1st to Kth shortest path lengths in given Graph Shortest path in a Binary Maze Minimum steps to reach target by a Knight Number of ways to reach at destination in shortest time Snake and Ladder Problem Word Ladder Comment Article Tags: Article Tags: Graph Dynamic Programming DSA Shortest Path