|  |  |
| --- | --- |
| **BellmanFord in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution {  public:  /\* Function to implement Bellman Ford  \* edges: vector of vectors which represents the graph  \* S: source vertex to start traversing graph with  \* V: number of vertices  \*/  vector<int> bellman\_ford(int V, vector<vector<int>>& edges, int S) {  vector<int> dist(V, 1e8);  dist[S] = 0;  for (int i = 0; i < V - 1; i++) {  for (auto it : edges) {  int u = it[0];  int v = it[1];  int wt = it[2];  if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {  dist[v] = dist[u] + wt;  }  }  }  // Nth relaxation to check negative cycle  for (auto it : edges) {  int u = it[0];  int v = it[1];  int wt = it[2];  if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {  return { -1};  }  }  return dist;  }  };  int main() {  int V = 6;  vector<vector<int>> edges(7, vector<int>(3));  edges[0] = {3, 2, 6};  edges[1] = {5, 3, 1};  edges[2] = {0, 1, 5};  edges[3] = {1, 5, -3};  edges[4] = {1, 2, -2};  edges[5] = {3, 4, -2};  edges[6] = {2, 4, 3};  int S = 0;  Solution obj;  vector<int> dist = obj.bellman\_ford(V, edges, S);  for (auto d : dist) {  cout << d << " ";  }  cout << endl;  return 0;  } | **Dry Run:**  Let's dry run the given code with the input:  int V = 6;  vector<vector<int>> edges(7, vector<int>(3));  edges[0] = {3, 2, 6};  edges[1] = {5, 3, 1};  edges[2] = {0, 1, 5};  edges[3] = {1, 5, -3};  edges[4] = {1, 2, -2};  edges[5] = {3, 4, -2};  edges[6] = {2, 4, 3};  int S = 0;  **Step 1: Initialize Variables**   * dist[]: Distance array initialized to {1e8, 1e8, 1e8, 1e8, 1e8, 1e8}. * Set dist[0] = 0 (since S = 0).   **Step 2: Relaxation (V-1) Times**   * **First iteration (i = 0)**: Relax all edges.   + Relax edge (3, 2, 6): No change.   + Relax edge (5, 3, 1): No change.   + Relax edge (0, 1, 5): dist[1] = min(1e8, dist[0] + 5) = 5.   + Relax edge (1, 5, -3): dist[5] = min(1e8, dist[1] - 3) = 2.   + Relax edge (1, 2, -2): dist[2] = min(1e8, dist[1] - 2) = 3.   + Relax edge (3, 4, -2): dist[4] = min(1e8, dist[3] - 2) = 3.   + Relax edge (2, 4, 3): No change. * **Second iteration (i = 1)**: Relax all edges again.   + Relax edge (3, 2, 6): No change.   + Relax edge (5, 3, 1): No change.   + Relax edge (0, 1, 5): No change.   + Relax edge (1, 5, -3): No change.   + Relax edge (1, 2, -2): No change.   + Relax edge (3, 4, -2): No change.   + Relax edge (2, 4, 3): No change.   (No updates during the second iteration.)   * **Third to Fifth iterations (i = 2, 3, 4)**: Relax all edges again.   + No further changes, as all shortest paths are already updated.   **Step 3: Negative Cycle Detection**   * **Nth iteration (i = 5)**: Perform one more relaxation round.   + All distances are unchanged, meaning no negative cycle exists.   **Step 4: Return the Result**   * Final dist[] array: {0, 5, 3, 3, 1, 2}.   Thus, the shortest distances from source 0 to all other nodes are:  0 5 3 3 1 2 |
| **Output:-**  0 5 3 3 1 2 | |