|  |  |
| --- | --- |
| **Kahn in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution {  public:      //Function to return list containing vertices in Topological order.      vector<int> topoSort(int V, vector<int> adj[])      {          int indegree[V] = {0};          for (int i = 0; i < V; i++) {              for (auto it : adj[i]) {                  indegree[it]++;              }          }          queue<int> q;          for (int i = 0; i < V; i++) {              if (indegree[i] == 0) {                  q.push(i);              }          }          vector<int> topo;          while (!q.empty()) {              int node = q.front();              q.pop();              topo.push\_back(node);              // node is in your topo sort              // so please remove it from the indegree              for (auto it : adj[node]) {                  indegree[it]--;                  if (indegree[it] == 0) q.push(it);              }          }          return topo;      }  };  int main() {      //V = 6;      vector<int> adj[6] = {{}, {}, {3}, {1}, {0, 1}, {0, 2}};      int V = 6;      Solution obj;      vector<int> ans = obj.topoSort(V, adj);      for (auto node : ans) {          cout << node << " ";      }      cout << endl;      return 0;  } | **Input:**  Graph:  4 ----> 0  | ^  | |  | +----+  v |  1 <--- 3 <---2  ^  |  5 ----> 2  |  +----> 0s  V = 6  Adjacency List:  0 -> {}  1 -> {}  2 -> {3}  3 -> {1}  4 -> {0, 1}  5 -> {0, 2}  **Step-by-Step Execution:**   1. **Calculate Indegree**:    * Traverse the adjacency list and compute indegrees:   Indegree of node 0 = 2 (edges from 4, 5)  Indegree of node 1 = 3 (edges from 3, 4, 5)  Indegree of node 2 = 1 (edge from 5)  Indegree of node 3 = 1 (edge from 2)  Indegree of node 4 = 0 (no incoming edges)  Indegree of node 5 = 0 (no incoming edges)   * + Indegree array: [2, 3, 1, 1, 0, 0]  1. **Initialize Queue**:    * Nodes with indegree = 0: [4, 5]    * Initial queue: q = [4, 5] 2. **Process Nodes in Topological Order**:    * **Step 1**: Process node 4:      + Add 4 to topo: topo = [4]      + Reduce indegree of 0 and 1: indegree[0] = 1, indegree[1] = 2      + Updated queue: q = [5]    * **Step 2**: Process node 5:      + Add 5 to topo: topo = [4, 5]      + Reduce indegree of 0 and 2: indegree[0] = 0, indegree[2] = 0      + Updated queue: q = [0, 2]    * **Step 3**: Process node 0:      + Add 0 to topo: topo = [4, 5, 0]      + No neighbors to update.      + Updated queue: q = [2]    * **Step 4**: Process node 2:      + Add 2 to topo: topo = [4, 5, 0, 2]      + Reduce indegree of 3: indegree[3] = 0      + Updated queue: q = [3]    * **Step 5**: Process node 3:      + Add 3 to topo: topo = [4, 5, 0, 2, 3]      + Reduce indegree of 1: indegree[1] = 0      + Updated queue: q = [1]    * **Step 6**: Process node 1:      + Add 1 to topo: topo = [4, 5, 0, 2, 3, 1]      + No neighbors to update.      + Updated queue: q = [] 3. **Final Topological Order**:   topo = [4, 5, 0, 2, 3, 1]  **Output:**  4 5 0 2 3 1 |
| **Output:-**  4 5 0 2 3 1 | |