|  |  |
| --- | --- |
| **Topological sort DFS in C++** | |
| #include <iostream>  #include <vector>  #include <stack>  using namespace std;  class Topo\_dfs {  public:  // Helper function to perform DFS and populate stack  static void dfs(int node, vector<int>& vis, stack<int>& st, vector<vector<int>>& adj) {  vis[node] = 1; // Mark node as visited  // Traverse all adjacent nodes  for (int it : adj[node]) {  if (vis[it] == 0) { // If adjacent node is not visited, perform DFS on it  dfs(it, vis, st, adj);  }  }  st.push(node); // Push current node to stack after visiting all its dependencies  }  // Function to perform topological sorting using DFS  static vector<int> topoSort(int V, vector<vector<int>>& adj) {  vector<int> vis(V, 0); // Initialize visited array  stack<int> st; // Stack to store nodes in topological order  // Perform DFS for each unvisited node  for (int i = 0; i < V; ++i) {  if (vis[i] == 0) {  dfs(i, vis, st, adj);  }  }  vector<int> topo(V);  int index = 0;  // Pop elements from stack to get topological order  while (!st.empty()) {  topo[index++] = st.top();  st.pop();  }  return topo;  }  };  int main() {  int V = 6;  vector<vector<int>> adj(V);  adj[2].push\_back(3);  adj[3].push\_back(1);  adj[4].push\_back(0);  adj[4].push\_back(1);  adj[5].push\_back(0);  adj[5].push\_back(2);  vector<int> ans = Topo\_dfs::topoSort(V, adj);  for (int node : ans) {  cout << node << " ";  }  cout << endl;  return 0;  } | **Input**  Vertices (V) = 6 Edges:   * 2 → 3 * 3 → 1 * 4 → 0 * 4 → 1 * 5 → 0 * 5 → 2   Adjacency list:  adj = [  [], // Node 0  [], // Node 1  [3], // Node 2  [1], // Node 3  [0, 1], // Node 4  [0, 2] // Node 5  ]  **Dry Run**  **Step 1: Initialize Variables**   * Visited array: vis = [0, 0, 0, 0, 0, 0] * Stack (st) is empty.   **Step 2: Start DFS from Unvisited Nodes**  **Iteration 1 (Node 0)**:   * vis[0] = 1. Node 0 has no neighbors. * Push 0 to st: st = [0].   **Iteration 2 (Node 1)**:   * vis[1] = 1. Node 1 has no neighbors. * Push 1 to st: st = [0, 1].   **Iteration 3 (Node 2)**:   * vis[2] = 1. * Neighbor: Node 3.   + Perform DFS on Node 3:     - vis[3] = 1.     - Neighbor: Node 1 (already visited).     - Push 3 to st: st = [0, 1, 3]. * Push 2 to st: st = [0, 1, 3, 2].   **Iteration 4 (Node 3)**:   * Already visited. Skip.   **Iteration 5 (Node 4)**:   * vis[4] = 1. * Neighbors: Node 0 and Node 1 (both already visited). * Push 4 to st: st = [0, 1, 3, 2, 4].   **Iteration 6 (Node 5)**:   * vis[5] = 1. * Neighbors: Node 0 and Node 2 (both already visited). * Push 5 to st: st = [0, 1, 3, 2, 4, 5].   **Step 3: Extract Topological Order**   * Reverse the stack: topo = [5, 4, 2, 3, 1, 0]. |
| **Output:-**  5 4 2 3 1 0 | |