|  |  |
| --- | --- |
| **Terminal Nodes in C++** | |
| #include <iostream>  #include <vector>  #include <unordered\_map>  #include <unordered\_set>  using namespace std;  class TerminalNodes {  private:  unordered\_map<int, vector<int>> adjacencyList;  public:  TerminalNodes() {}  void addEdge(int source, int destination) {  adjacencyList[source].push\_back(destination);  adjacencyList[destination]; // Ensure destination is also in the map  }  void printTerminalNodes() {  vector<int> terminalNodes;  for (auto it = adjacencyList.begin(); it != adjacencyList.end(); ++it) {  if (it->second.empty()) {  terminalNodes.push\_back(it->first);  }  }  cout << "Terminal Nodes:" << endl;  for (int node : terminalNodes) {  cout << node << endl;  }  }  };  int main() {  TerminalNodes graph;  // Adding edges to the graph  graph.addEdge(1, 2);  graph.addEdge(2, 3);  graph.addEdge(3, 4);  graph.addEdge(4, 5);  graph.addEdge(6, 7);  graph.printTerminalNodes();  return 0;  } | ****Step-by-Step Dry Run****  | **Step** | **Operation** | **Affected Node(s)** | **Adjacency List State** | **Notes** | | --- | --- | --- | --- | --- | | 1 | addEdge(1, 2) | 1, 2 | {1: [2], 2: []} | 1 → 2, ensure 2 is in the map | | 2 | addEdge(2, 3) | 2, 3 | {1: [2], 2: [3], 3: []} | 2 → 3, ensure 3 is in the map | | 3 | addEdge(3, 4) | 3, 4 | {1: [2], 2: [3], 3: [4], 4: []} | 3 → 4, ensure 4 is in the map | | 4 | addEdge(4, 5) | 4, 5 | {1: [2], 2: [3], 3: [4], 4: [5], 5: []} | 4 → 5, ensure 5 is in the map | | 5 | addEdge(6, 7) | 6, 7 | {1: [2], 2: [3], 3: [4], 4: [5], 5: [], 6: [7], 7: []} | 6 → 7, ensure 7 is in the map | | 6 | printTerminalNodes() | Scan all nodes | Check which nodes have empty adjacency lists | Nodes 5 and 7 have no outgoing edges | | 7 | Print | Terminal Nodes |  | Output: 5, 7 |  ✅ ****Final Output****Terminal Nodes: 5  7 |
| **Output:-**  Terminal Nodes:  7  5 | |