|  |  |
| --- | --- |
| **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;  } | **Example Walkthrough**  Let's consider the following graph representation:  1 -> 2 -> 3 -> 4 -> 5  6 -> 7   * **Graph Representation:**   + Node 1 has an edge to node 2.   + Node 2 has an edge to node 3.   + Node 3 has an edge to node 4.   + Node 4 has an edge to node 5.   + Node 6 has an edge to node 7.   + Node 7 has no outgoing edges. * **Terminal Nodes:**   + Nodes 5 and 7 are terminal nodes because they have no outgoing edges.   **Code Execution:**   1. The addEdge method is called multiple times to build the graph. 2. Then, the printTerminalNodes() method is called to iterate through the graph and check for terminal nodes. 3. The nodes 5 and 7 will be identified as terminal nodes and printed. |
| **Output:-**  Terminal Nodes:  7  5 | |