```
#include <iostream>A
#include <vector>
#include <stack>

using namespace std;

class Graph {
  int V;# no. of vertex
  vector<int>* adj;# array of vectors
  int time;
why?? stack<int> Stack;
  vector<int> ids, low, inStack;

  void sccDFS(int v);

public:
  Graph(int V);
  void addEdge(int v, int w);# v → w
  void scc();
};

Graph::Graph(int V) {
    this->V = V;
    adj = new vector<int>[V];
    ids = vector<int>(V, -1);# -1 denote unvisited
    low = vector<int>(V, -1);
    inStack = vector<int>(V, 0);# bool type
    time = 0;
}

void Graph::addEdge(int v, int w) {
    adj[v].push_back(w);
}

void Graph::sccDFS(int u) {
    ids[u] = low[u] = time++;
    Stack.push(u);              } → Start
    inStack[u] = 1;

    for (int v : adj[u]) {
        if (ids[v] == -1) {
          sccDFS(v); // backtrack from v's exploration
            low[u] = min(low[u], low[v]);
        } else if (inStack[v]) {
            low[u] = min(low[u], ids[v]);
            /*
              v.lowlink := min(v.lowlink, w.index) is the correct
way
              to update v.lowlink if w is on stack. Because w is on
              the stack already, (v, w) is a back-edge in the DFS
tree
              and therefore w is not in the subtree of v. Because
              v.lowlink takes into account nodes reachable only
              through the nodes in the subtree of v we must stop at
```

At the end of loop, lowlink of u is updated based on if its neighbour w is yet to be explored OR if w is explored, but doesnt belong to scc

```
w
                and use w.index instead of w.lowlink

               */
        }
    } // for loop for LL value computation has ended here.

    // SCC computation begins
    // I am in DFS(u)
    int w = 0;
    // u is the start of an SCC
    // Print the SCC and remove them from the stack
    if (low[u] == ids[u]) {
        while (Stack.top() != u) {
            w = Stack.top();
            cout << w << " ";
            inStack[w] = 0;
            Stack.pop();
        }
        w = Stack.top();
        cout << w << "\n";
        inStack[w] = 0;
        Stack.pop();
    }
}
    // Convince yourself of the correctness iof this LL update
mechanism}
    // Find the loop invariants for the SCCDFS computation

void Graph::scc() {
    for (int i = 0; i < V; i++) {
        if (ids[i] == -1) {
            sccDFS(i);
        }
    }
}

int main() {
    // Example usage:
    Graph g(5);
    g.addEdge(1, 0);
    g.addEdge(0, 2);
    g.addEdge(2, 1);
    g.addEdge(0, 3);
    g.addEdge(3, 4);
    g.scc();

    return 0;
}
```

if u.id == u.lowlink

→ all element above u are in same SCC