```
#ifndef LAB_7_GRAPH_H
#define LAB_7_GRAPH_H

#include <iostream>
#include <vector>
#include <list>
using std::ostream;
using std::vector;
using std::list;
class Graph {
public:
    explicit Graph(int V) : v(V), e(0), adjlist(V){ }
    int V() const;

    int E() const;

    void addEdge(int v, int w);

    const list<int>& adj(int v) const;

    friend ostream &operator<<(ostream &os, const Graph &graph);


private:
    vector<list<int>> adjlist;
    int v;
    int e;
};


#endif //LAB_7_GRAPH_H

#include "Graph.h"

int Graph::V() const {
    return v;
}

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

ostream& operator<<(ostream& os, const Graph& graph){
    os << "Vertices: " << graph.V() << " edges: " << graph.E() << std::endl;
    for(int v = 0;v < graph.V();v++){
        os << v << " : {";
        for(int w : graph.adj(v)){
            os << w << " ";
        }
        os << "}" << std::endl;
    }
    return os;
}

const list<int>& Graph::adj(int v) const {
    return adjlist[v];
}
```

```
int Graph::E() const {
    return e;
}

#ifndef LAB8_CYCLEDETECTOR_H
#define LAB8_CYCLEDETECTOR_H
#include "Graph.h"
#include <iostream>
#include <limits>

class CycleDetector {
public:
    explicit CycleDetector(const Graph& G);
    bool has_cycle();
private:
    void dfs(const Graph& G, int v);
private:
    vector<int> start_times;
    vector<int> end_times;
    vector<int> parent;
    bool cycle_found;
    int timer;
};


#endif //LAB8_CYCLEDETECTOR_H

//
// Created by sergio on 12/2/20.
//

#include "CycleDetector.h"

CycleDetector::CycleDetector(const Graph &G) : parent(G.V(),
std::numeric_limits<int>::lowest())
,start_times(G.V(), -1), end_times(G.V(), -1), timer(0), cycle_found(false)
{
    for(int v = 0;v < G.V();v++){
        if(parent[v] == std::numeric_limits<int>::lowest()){
            parent[v] = -1;
            dfs(G, v);
        }
    }
}

bool CycleDetector::has_cycle() {
    return cycle_found;
}

void CycleDetector::dfs(const Graph &G, int v) {
    start_times[v] = ++timer;
    for(int w : G.adj(v)){
        if(parent[w] == std::numeric_limits<int>::lowest()){
            parent[w] = v;
            dfs(G, w);
        } else if(end_times[w] == -1){
            cycle_found = true;
            std::cout << "Cycle detected, topological sort is impossible" <<
std::endl;
        }
    }
    end_times[v] = ++timer;
}


#ifndef LAB8_TOPOLOGICALSORT_H
#define LAB8_TOPOLOGICALSORT_H
#include "Graph.h"
#include "CycleDetector.h"

class TopologicalSort {
public:
    explicit TopologicalSort(const Graph& G);
    int start_time(int v);
    int end_time(int v);
    const vector<int>& topological_order() const;
private:
    void dfs(const Graph& G, int v);
    vector<int> start_times;
    vector<int> end_times;
    vector<int> parent;
    vector<int> top_order;
    int timer;
};


#endif //LAB8_TOPOLOGICALSORT_H

#include "TopologicalSort.h"
#include <algorithm>

TopologicalSort::TopologicalSort(const Graph &G)
: parent(G.V(), std::numeric_limits<int>::lowest())
,start_times(G.V(), -1), end_times(G.V(), -1), timer(0)
{
    for(int v = 0;v < G.V();v++){
        if(parent[v] == std::numeric_limits<int>::lowest()){
            parent[v] = -1;
            dfs(G, v);
        }
    }
    std::reverse(top_order.begin(), top_order.end());
}

void TopologicalSort::dfs(const Graph &G, int v) {
    start_times[v] = ++timer;
    for(int w : G.adj(v)){
        if(parent[w] == std::numeric_limits<int>::lowest()){
            parent[w] = v;
            dfs(G, w);
        }
    }
    end_times[v] = ++timer;
    top_order.push_back(v);
}

int TopologicalSort::start_time(int v) {
    return start_times[v];
}

int TopologicalSort::end_time(int v) {
    return end_times[v];
}

const vector<int> &TopologicalSort::topological_order() const {
    return top_order;
}

#include <iostream>
#include <sstream>
#include <random>
#include <chrono>
#include <limits>
#include "Graph.h"
#include "CycleDetector.h"
#include "TopologicalSort.h"

using namespace std;

bool getline(const string& prompt, string& userinput){
    cout << prompt;
    getline(cin, userinput);
    return  !userinput.empty();
}

Graph generate_graph(int V, int E){
    vector<pair<int, int>> all_subsets;
    for(int i = 0;i < V;i++){
        for(int j = 0;j < V;j++){
            if(i != j){
                all_subsets.push_back({i, j});
            }
        }
    }
    long seed = chrono::steady_clock::now().time_since_epoch().count();
    mt19937 gen(seed);
    uniform_int_distribution<int> uniformIntDistribution(0, E);
    vector<pair<int, int>> random_subset;
    for(int i = 0;i < E;i++){
        random_subset.push_back(all_subsets[i]);
    }
    for(int i = E;i < all_subsets.size();i++){
        int random_idx = uniformIntDistribution(gen);
        if(random_idx < E){
            random_subset[random_idx] = all_subsets[i];
        }
    }
    Graph G(V);
    for(const auto& p : random_subset){
        G.addEdge(p.first, p.second);
    }
    return G;
}

int main() {
    string userinput;
    while(getline("Enter the number vertices followed by the number of edges: ",
userinput)){
        int V, E;
        stringstream ss(userinput);
        ss >> V >> E;
        Graph G = generate_graph(V, E);
        cout << G << endl;
        CycleDetector cycleDetector(G);
        if(!cycleDetector.has_cycle()){
            TopologicalSort topologicalSort(G);
            for(int v : topologicalSort.topological_order()){
                cout << "v" << v << " start time: " <<
topologicalSort.start_time(v) << " end time: " << topologicalSort.end_time(v) <<
endl;
            }
        }
    }
}
```

```
/home/sergio/Desktop/lab8/cmake-build-debug/lab8
Enter the number vertices followed by the number of edges: 5 3
Vertices: 5 edges: 3
0 : {}
1 : {}
2 : {}
3 : {}
4 : {1 3 0 }

v4 start time: 9 end time: 10
v3 start time: 7 end time: 8
v2 start time: 5 end time: 6
v1 start time: 3 end time: 4
v0 start time: 1 end time: 2
Enter the number vertices followed by the number of edges: 5 12
Vertices: 5 edges: 12
0 : {1 2 3 }
1 : {2 }
2 : {1 3 4 }
3 : {2 1 }
4 : {2 3 0 }

Cycle detected, topological sort is impossible
Cycle detected, topological sort is impossible
Cycle detected, topological sort is impossible
Cycle detected, topological sort is impossible
Cycle detected, topological sort is impossible
Enter the number vertices followed by the number of edges: 5 10
Vertices: 5 edges: 10
0 : {1 3 4 }
1 : {2 }
2 : {}
3 : {2 4 1 }
4 : {2 3 1 }

Cycle detected, topological sort is impossible
Enter the number vertices followed by the number of edges: 5 5
Vertices: 5 edges: 5
0 : {}
1 : {}
```

```
2 : {}
3 : {4 1 }
4 : {1 2 3 }

Cycle detected, topological sort is impossible
Enter the number vertices followed by the number of edges: 5 5
Vertices: 5 edges: 5
0 : {}
1 : {3 2 }
2 : {}
3 : {4 1 }
4 : {1 }

Cycle detected, topological sort is impossible
Cycle detected, topological sort is impossible
Enter the number vertices followed by the number of edges: 5 2
Vertices: 5 edges: 2
0 : {}
1 : {}
2 : {}
3 : {}
4 : {3 1 }

v4 start time: 9 end time: 10
v3 start time: 7 end time: 8
v2 start time: 5 end time: 6
v1 start time: 3 end time: 4
v0 start time: 1 end time: 2
```