```
 1
 2 #ifndef LAB_7_GRAPH_H
 3 #define LAB_7_GRAPH_H
 4
 5 #include <iostream>
 6 #include <vector>
 7 #include <list>
 8 using std::ostream;
 9 using std::vector;
10 using std::list;
11 class Graph {
12 public:
13     explicit Graph(int V) : v(V), e(0), adjlist(V){ }
14     int V() const;
15
16     int E() const;
17
18     void addEdge(int v, int w);
19
20     const list<int>& adj(int v) const;
21
22     friend ostream &operator<<(ostream &os, const Graph &
   graph);
23
24
25 private:
26     vector<list<int>> adjlist;
27     int v;
28     int e;
29 };
30
31
32 #endif //LAB_7_GRAPH_H
```

File - /home/sergio/Desktop/lab8/main.cpp

```cpp
#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;
```

```
49      while(getline("Enter the number vertices followed by
    the number of edges: ", userinput)){
50          int V, E;
51          stringstream ss(userinput);
52          ss >> V >> E;
53          Graph G = generate_graph(V, E);
54          cout << G << endl;
55          CycleDetector cycleDetector(G);
56          if(!cycleDetector.has_cycle()){
57              TopologicalSort topologicalSort(G);
58              for(int v : topologicalSort.topological_order(
    )){
59                  cout << "v" << v << " start time: " <<
    topologicalSort.start_time(v) << " end time: " <<
    topologicalSort.end_time(v) << endl;
60              }
61          }
62      }
63  }
```

```cpp
#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;
}


```

File - /home/sergio/Desktop/lab8/CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.12)
2 project(lab8)
3
4 set(CMAKE_CXX_STANDARD 14)
5
6 add_executable(lab8 main.cpp Graph.cpp Graph.h
  CycleDetector.cpp CycleDetector.h TopologicalSort.cpp
  TopologicalSort.h)
```

```h
#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

```

```cpp
//
// 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(parent[w] != std::numeric_limits<int>::lowest() && end_times[w] == -1){
            cycle_found = true;
            std::cout << "Cycle detected, topological sort is impossible" << std::endl;
        }
    }
    end_times[v] = timer++;
}

```

```h
#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

```

```cpp
#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;
}

```