

```
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
```

```

1  #include <iostream>
2  #include <sstream>
3  #include <random>
4  #include <chrono>
5  #include <limits>
6  #include "Graph.h"
7  #include "CycleDetector.h"
8  #include "TopologicalSort.h"
9
10 using namespace std;
11
12 bool getline(const string& prompt, string& userinput){
13     cout << prompt;
14     getline(cin, userinput);
15     return !userinput.empty();
16 }
17
18 Graph generate_graph(int V, int E){
19     vector<pair<int, int>> all_subsets;
20     for(int i = 0; i < V; i++){
21         for(int j = 0; j < V; j++){
22             if(i != j){
23                 all_subsets.push_back({i, j});
24             }
25         }
26     }
27     long seed = chrono::steady_clock::now().
time_since_epoch().count();
28     mt1937 gen(seed);
29     uniform_int_distribution<int> uniformIntDistribution(0
, E);
30     vector<pair<int, int>> random_subset;
31     for(int i = 0; i < E; i++){
32         random_subset.push_back(all_subsets[i]);
33     }
34     for(int i = E; i < all_subsets.size(); i++){
35         int random_idx = uniformIntDistribution(gen);
36         if(random_idx < E){
37             random_subset[random_idx] = all_subsets[i];
38         }
39     }
40     Graph G(V);
41     for(const auto& p : random_subset){
42         G.addEdge(p.first, p.second);
43     }
44     return G;
45 }
46
47 int main() {
48     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(  
60                 )){  
59                 cout << "v" << v << " start time: " <<  
topologicalSort.start_time(v) << " end time: " <<  
topologicalSort.end_time(v) << endl;  
60                 }  
61             }  
62         }  
63     }
```

```
1 #include "Graph.h"
2
3 int Graph::V() const {
4     return v;
5 }
6
7 void Graph::addEdge(int v, int w) {
8     adjlist[v].push_back(w);
9     e++;
10 }
11
12 ostream& operator<<(ostream& os, const Graph& graph){
13     os << "Vertices: " << graph.V() << " edges: " << graph
14     .E() << std::endl;
15     for(int v = 0; v < graph.V(); v++){
16         os << v << " : {";
17         for(int w : graph.adj(v)){
18             os << w << " ";
19         }
20         os << "}" << std::endl;
21     }
22     return os;
23 }
24
25 const list<int>& Graph::adj(int v) const {
26     return adjlist[v];
27 }
28
29 int Graph::E() const {
30     return e;
31 }
32
33
```

```
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)
```

```
1 #ifndef LAB8_CYCLEDETECTOR_H
2 #define LAB8_CYCLEDETECTOR_H
3 #include "Graph.h"
4 #include <iostream>
5 #include <limits>
6
7 class CycleDetector {
8 public:
9     explicit CycleDetector(const Graph& G);
10    bool has_cycle();
11 private:
12    void dfs(const Graph& G, int v);
13 private:
14    vector<int> start_times;
15    vector<int> end_times;
16    vector<int> parent;
17    bool cycle_found;
18    int timer;
19 };
20
21
22 #endif //LAB8_CYCLEDETECTOR_H
23
```

```

1 //
2 // Created by sergio on 12/2/20.
3 //
4
5 #include "CycleDetector.h"
6
7 CycleDetector::CycleDetector(const Graph &G) : parent(G.V(
8 ), std::numeric_limits<int>::lowest())
9 , start_times(G.V(), -1), end_times(G.V(), -1), timer(0),
10 cycle_found(false)
11 {
12     for(int v = 0; v < G.V(); v++){
13         if(parent[v] == std::numeric_limits<int>::lowest())
14         ){
15             parent[v] = -1;
16             dfs(G, v);
17         }
18     }
19 }
20
21 bool CycleDetector::has_cycle() {
22     return cycle_found;
23 }
24
25 void CycleDetector::dfs(const Graph &G, int v) {
26     start_times[v] = ++timer;
27     for(int w : G.adj(v)){
28         if(parent[w] == std::numeric_limits<int>::lowest())
29         ){
30             parent[w] = v;
31             dfs(G, w);
32         } else if(end_times[w] == -1){
33             cycle_found = true;
34             std::cout << "Cycle detected, topological sort
35 is impossible" << std::endl;
36         }
37     }
38     end_times[v] = ++timer;
39 }
40

```

```
1 #ifndef LAB8_TOPOLOGICALSORT_H
2 #define LAB8_TOPOLOGICALSORT_H
3 #include "Graph.h"
4 #include "CycleDetector.h"
5
6 class TopologicalSort {
7 public:
8     explicit TopologicalSort(const Graph& G);
9     int start_time(int v);
10    int end_time(int v);
11    const vector<int>& topological_order() const;
12 private:
13     void dfs(const Graph& G, int v);
14     vector<int> start_times;
15     vector<int> end_times;
16     vector<int> parent;
17     vector<int> top_order;
18     int timer;
19 };
20
21
22 #endif //LAB8_TOPOLOGICALSORT_H
23
```



```

1  #include "TopologicalSort.h"
2  #include <algorithm>
3
4  TopologicalSort::TopologicalSort(const Graph &G)
5  : parent(G.V(), std::numeric_limits<int>::lowest())
6  , start_times(G.V(), -1), end_times(G.V(), -1), timer(0)
7  {
8      for(int v = 0; v < G.V(); v++){
9          if(parent[v] == std::numeric_limits<int>::lowest())
10         ){
11             parent[v] = -1;
12             dfs(G, v);
13         }
14     }
15     std::reverse(top_order.begin(), top_order.end());
16 }
17 void TopologicalSort::dfs(const Graph &G, int v) {
18     start_times[v] = ++timer;
19     for(int w : G.adj(v)){
20         if(parent[w] == std::numeric_limits<int>::lowest())
21         ){
22             parent[w] = v;
23             dfs(G, w);
24         }
25     }
26     end_times[v] = ++timer;
27     top_order.push_back(v);
28 }
29 int TopologicalSort::start_time(int v) {
30     return start_times[v];
31 }
32
33 int TopologicalSort::end_time(int v) {
34     return end_times[v];
35 }
36
37 const vector<int> &TopologicalSort::topological_order()
38 const {
39     return top_order;
40 }

```