File - /home/sergio/Desktop/lab_7/Graph.h

```
 1 #ifndef LAB_7_GRAPH_H
 2 #define LAB_7_GRAPH_H
 3
 4 #include <iostream>
 5 #include <vector>
 6 #include <list>
 7 using std::ostream;
 8 using std::vector;
 9 using std::list;
10
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
33
```

File - /home/sergio/Desktop/lab_7/main.cpp

```cpp
#include <iostream>
#include <limits>
#include <sstream>
#include <deque>
#include <chrono>
#include <queue>
#include <random>
#include "Graph.h"

using namespace std;

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

void bfs(const Graph& G, int source){
    vector<int> distTo(G.V(), std::numeric_limits<int>::max());
    deque<int> edgeTo(G.V(), -1);
    distTo[source] = 0;
    edgeTo[source] = std::numeric_limits<int>::lowest();
    queue<int> q;
    q.push(source);
    while(!q.empty()){
        int v = q.front();
        q.pop();
        for(int w : G.adj(v)){
            if(edgeTo[w] == -1){
                edgeTo[w] = v;
                distTo[w] = distTo[v] + 1;
                q.push(w);
            }
        }
    }
    for(int v = 0;v < G.V();v++){
        if(distTo[v] != std::numeric_limits<int>::max()){
            cout << "Shortest Path cost from: "<< source << " to " <<  v << " is " << distTo[v] << endl;
            vector<int> path;
            for(int e = v;e != std::numeric_limits<int>::lowest();e = edgeTo[e]){
                path.push_back(e);
            }
            for(int i = path.size() - 1;i >= 1;i--){
                cout << path[i] << "->";
            }
            cout << path[0] << endl;
        } else {
```

```
48              cout << source << " to " << v << " unreachable
   " << endl;
49          }
50      }
51  }
52
53  enum COLORS{GRAY = 0, RED = 1, BLUE = 2};
54
55  bool is_bipartite(const Graph& G, int source, vector<
   COLORS>& colors){
56      colors[source] = BLUE;
57      queue<int> q;
58      q.push(source);
59      while(!q.empty()){
60          int v = q.front();
61          q.pop();
62          for(int w : G.adj(v)){
63              if(colors[w] == GRAY){
64                  colors[w] = (colors[v] == RED) ? BLUE :
   RED;
65                  q.push(w);
66              } else if(colors[w] == colors[v]){
67                  cout << "not bipartite" << endl;
68                  return false;
69              }
70          }
71      }
72      return true;
73  }
74
75  void explore(const Graph& G){
76      vector<COLORS > vertex_color(G.V(), GRAY);
77      for(int v = 0;v < G.V();v++){
78          if(vertex_color[v] == GRAY && !is_bipartite(G, v,
   vertex_color)){
79              break;
80          }
81      }
82      vector<string> color_decoded = {"gray", "red", "blue"}
   ;
83      for(int v = 0;v < G.V();v++){
84          cout << v << " color: " << color_decoded[
   vertex_color[v]] << endl;
85      }
86  }
87
88  Graph generate_graph(int V, int E){
89      vector<pair<int, int>> all_subsets;
90      for(int i = 0;i < V;i++){
91          for(int j = i + 1;j < V;++j){
```

```
 92                 all_subsets.push_back({i, j});
 93             }
 94         }
 95         vector<pair<int, int>> subset;
 96         for(int i = 0;i < E;i++){
 97             subset.push_back(all_subsets[i]);
 98         }
 99         long seed = chrono::system_clock::now().
    time_since_epoch().count();
100         mt19937 gen(seed);
101         uniform_int_distribution<int>
    uniform_int_distribution(0, E - 1);
102         for(int i = E;i < all_subsets.size();i++){
103             int random_idx = uniform_int_distribution(gen);
104             subset[random_idx] = all_subsets[i];
105         }
106         Graph G(V);
107         for(const auto& p : subset){
108             G.addEdge(p.first, p.second);
109         }
110         return G;
111 }
112 
113 int main() {
114         string userinput;
115         while(get_line("(part a) enter number of vertices
    followed by number of edges separated by a space: ",
    userinput)){
116             stringstream ss(userinput);
117             int V, E;
118             ss >> V >> E;
119             Graph G = generate_graph(V, E);
120             cout << G << endl;
121             get_line("enter starting vertex of bfs: ",
    userinput);
122             ss = stringstream(userinput);
123             int source;
124             ss >> source;
125             bfs(G, source);
126         } while(get_line("(part b) enter number of vertices
    followed by number of edges separated by a space: ",
    userinput)){
127             stringstream ss(userinput);
128             int V, E;
129             ss >> V >> E;
130             Graph G = generate_graph(V, E);
131             cout << G << endl;
132             explore(G);
133         }
134 }
```

```cpp
#include "Graph.h"

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

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;
    }
    os << std::endl;
    return os;
}

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

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

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

```

File - /home/sergio/Desktop/lab_7/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.12)
project(lab_7)

set(CMAKE_CXX_STANDARD 14)

add_executable(lab_7 main.cpp Graph.cpp Graph.h)
```