```
#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;
}

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];
}

#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] = source;
    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 != source;e = edgeTo[e]){
                path.push_back(e);
            }
            path.push_back(source);
            for(int i = path.size() - 1;i >= 1;i--){
                cout << path[i] << "->";
            }
            cout << path[0] << endl;
        } else {
            cout << source << " to " << v << " unreachable" << endl;
        }
    }
}

enum COLORS{GRAY = 0, RED = 1, BLUE = 2};

bool is_bipartite(const Graph& G, int source, vector<COLORS>& colors){
    colors[source] = BLUE;
    queue<int> q;
    q.push(source);
    while(!q.empty()){
        int v = q.front();
        q.pop();
        for(int w : G.adj(v)){
            if(colors[w] == GRAY){
                colors[w] = (colors[v] == RED) ? BLUE : RED;
                q.push(w);
            } else if(colors[w] == colors[v]){
                cout << "not bipartite" << endl;
                return false;
            }
        }
    }
    return true;
}

void explore(const Graph& G){
    vector<COLORS > vertex_color(G.V(), GRAY);
    for(int v = 0;v < G.V();v++){
        if(vertex_color[v] == GRAY && !is_bipartite(G, v, vertex_color)){
            break;
        }
    }
    vector<string> color_decoded = {"gray", "red", "blue"};
    for(int v = 0;v < G.V();v++){
        cout << v << " color: " << color_decoded[vertex_color[v]] << endl;
    }
}

Graph generate_graph(int V, int E){
    vector<pair<int, int>> all_subsets;
    for(int i = 0;i < V;i++){
        for(int j = i + 1;j < V;++j){
            all_subsets.push_back({i, j});
        }
    }
    vector<pair<int, int>> subset;
    for(int i = 0;i < E;i++){
        subset.push_back(all_subsets[i]);
    }
    long seed = chrono::system_clock::now().time_since_epoch().count();
    mt19937 gen(seed);
    uniform_int_distribution<int> uniform_int_distribution(0, E);
    for(int i = E;i < all_subsets.size();i++){
        int random_idx = uniform_int_distribution(gen);
        if(random_idx < E) {
            subset[random_idx] = all_subsets[i];
        }
    }
    Graph G(V);
    for(const auto& p : subset){
        G.addEdge(p.first, p.second);
    }
    return G;
}

int main() {
    string userinput;
    while(get_line("(part a) enter number of vertices followed by number of edges separated by a space: ", userinput))
{
        stringstream ss(userinput);
        int V, E;
        ss >> V >> E;
        Graph G = generate_graph(V, E);
        cout << G << endl;
        get_line("enter starting vertex of bfs: ", userinput);
        ss = stringstream(userinput);
        int source;
        ss >> source;
        bfs(G, source);
    } while(get_line("(part b) enter number of vertices followed by number of edges separated by a space: ", userinput
)){
        stringstream ss(userinput);
        int V, E;
        ss >> V >> E;
        Graph G = generate_graph(V, E);
        cout << G << endl;
        explore(G);
    }
}
```

```
(part a) enter number of vertices followed by number of edges separated by a space: 5 7
Vertices: 5 edges: 7
0 : {1 3 4 }
1 : {0 3 4 }
2 : {3 }
3 : {2 0 4 1 }
4 : {0 3 1 }
```

```
enter starting vertex of bfs: 3
Shortest Path cost from: 3 to 0 is 1
3->0
Shortest Path cost from: 3 to 1 is 1
3->1
Shortest Path cost from: 3 to 2 is 1
3->2
Shortest Path cost from: 3 to 3 is 0
3
Shortest Path cost from: 3 to 4 is 1
3->4
(part a) enter number of vertices followed by number of edges separated by a space: 5 10
Vertices: 5 edges: 10
0 : {1 2 3 4 }
1 : {0 2 3 4 }
2 : {0 1 3 4 }
3 : {0 1 2 4 }
4 : {0 1 2 3 }


enter starting vertex of bfs: 1
Shortest Path cost from: 1 to 0 is 1
1->0
Shortest Path cost from: 1 to 1 is 0
1
Shortest Path cost from: 1 to 2 is 1
1->2
Shortest Path cost from: 1 to 3 is 1
1->3
Shortest Path cost from: 1 to 4 is 1
1->4

/home/sergio/Desktop/lab_7/cmake-build-debug/lab_7
(part a) enter number of vertices followed by number of edges separated by a space:
(part b) enter number of vertices followed by number of edges separated by a space: 5 5
Vertices: 5 edges: 5
0 : {1 3 }
1 : {0 4 }
2 : {3 }
3 : {0 2 4 }
4 : {1 3 }


0 color: blue
1 color: red
2 color: blue
3 color: red
4 color: blue
(part b) enter number of vertices followed by number of edges separated by a space: 5 10
Vertices: 5 edges: 10
0 : {1 2 3 4 }
1 : {0 2 3 4 }
2 : {0 1 3 4 }
3 : {0 1 2 4 }
```

```
4 : {0 1 2 3 }


not bipartite
0 color: blue
1 color: red
2 color: red
3 color: red
4 color: red
```