Practical 2

18BCE243

Code of Practical 2

```
#include <set>
#include <algorithm>
#include <stdexcept>
#include <map>
#include <vector>

template<typename T>
class UndirectedGraph {
public:
    std::set<T> V;
    std::set<std::pair<T, T> > E;

    UndirectedGraph() {}
    UndirectedGraph(std::set<T> _V, std::set<std::pair<T, T> > _E)
    {
        for ( auto it = _E.begin() ; it != _E.end() ; ++it ) {
            auto v1 = it->first;
            auto v2 = it->second;

            if ( _V.find(v1) == _V.end() || _V.find(v2) == _V.end() ) {
                throw std::logic_error("error: The edge set contains"\
                                       "vertices not present in the "\
                                       "vertex set!");
            }
        }

        V = _V;
        E = _E;
    }

    std::vector<std::pair<T, T> >
    check_isomorphism(UndirectedGraph &G2) {
        std::vector<std::pair<T, T> > dummy;
        std::set<T> V1 = this->V;
        std::set<T> V2 = G2.V;

        std::set<std::pair<T, T> > E1 = this->E;
        std::set<std::pair<T, T> > E2 = G2.E;

        bool conditions = ((V1.size() == V2.size()) && (E1.size() == E2.size()));
```

```
    if ( !conditions ) return dummy;
    else {
        std::map<T, std::set<T> > G1_adj_list = this->get_adj_list();
        std::vector<std::pair<T, int> > G1_degrees = \
            this->get_degrees_of_vertices(G1_adj_list);

        std::sort(G1_degrees.begin(), G1_degrees.end(),
            [](std::pair<T, int> a, std::pair<T, int> b) {
                return (a.second < b.second);
            }
        );

        std::map<T, std::set<T> > G2_adj_list = G2.get_adj_list();
        std::vector<std::pair<T, int> > G2_degrees = \
            G2.get_degrees_of_vertices(G2_adj_list);

        std::sort(G2_degrees.begin(), G2_degrees.end(),
            [](std::pair<T, int> a, std::pair<T, int> b) {
                return (a.second < b.second);
            }
        );

        for (size_t i=0 ; i<G1_degrees.size() ; ++i ) {
            if ( G1_degrees[i].second != G2_degrees[i].second ) return dummy;
        }

        std::vector<std::pair<T, T> > result;

        for (size_t i=0 ; i<G1_degrees.size() ; ++i ) {
            result.push_back(std::make_pair(G1_degrees[i].first,
                                            G2_degrees[i].first));
        }

        return result;
    }
}

std::vector<std::pair<T, int> >
get_degrees_of_vertices(const std::map<T, std::set<T> > &adj_list) {
    std::vector<std::pair<T, int> > degrees;
    for (const auto &p : adj_list) {
        auto vert = p.first;
        auto adj = p.second;
        degrees.push_back(std::make_pair(vert, adj.size()));
    }
```

```
        return degrees;
    }

    std::map<T, std::set<T> >
    get_adj_list() {
        std::map<T, std::set<T> > adj_list;
        for ( auto it = this->E.begin() ; it != this->E.end() ; ++it ) {
            adj_list[it->first].insert(it->second);
            adj_list[it->second].insert(it->first);
        }
        return adj_list;
    }

    friend std::ostream& operator<<(std::ostream &fout, const UndirectedGraph &_G)
    {
        fout << "Vertex Set: { ";
        for ( auto it = _G.V.begin() ; it != _G.V.end() ; ++it ) {
            fout << *it << " ";
        }
        fout << "}\n";
        fout << "Edge Set: { ";
        for ( auto it = _G.E.begin() ; it != _G.E.end() ; ++it ) {
            fout << "{" << it->first << ", " << it->second << "} ";
        }
        fout << "}";

        return fout;
    }
};
```

Test Driver (with Inputs)

```
#include <iostream>
#include "practical2.h"

int main()
{
    /*

    G1 -->  p --- r
            |     |
            |     |
            q --- s --- t

    G2 -->  a --- c
            |     |
```

```
        |     |
        d --- e --- b

    Given Graphs are Isomorphic with similar vertices:

    p --- a
    r --- c  OR  r --- d
    q --- d  OR  q --- c
    s --- e
    t --- b

    */

    std::set<char> V1 = {'p', 'q', 'r', 's', 't'};
    std::set<std::pair<char, char> > E1 = {{'p', 'q'},
                                           {'p', 'r'},
                                           {'r', 's'},
                                           {'q', 's'},
                                           {'s', 't'}};
    auto G1 = UndirectedGraph<char>(V1, E1);

    std::set<char> V2 = {'a', 'b', 'c', 'd', 'e'};
    std::set<std::pair<char, char> > E2 = {{'a', 'c'},
                                           {'a', 'd'},
                                           {'c', 'e'},
                                           {'d', 'e'},
                                           {'e', 'b'}};
    auto G2 = UndirectedGraph<char>(V2, E2);

    auto result = G1.check_isomorphism(G2);

    if ( result.empty() ) {
        std::cout << "Given graphs are not isomorphic!!" << std::endl;
    }
    else {
        std::cout << "Given graphs are isomorphic!\n" << std::endl;
        std::cout << "Similar Vertices\n----------------" << std::endl;
        for ( const auto &verts : result ) {
            std::cout << "{" << verts.first << ", " << verts.second << "}" << std::endl;
        }
    }

    return 0;
}
```

Output

```
Given graphs are isomorphic!

Similar Vertices
----------------
{t, b}
{p, a}
{q, c}
{r, d}
{s, e}
```