```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * File:   IGraph.h
 * Author: LTSACH
 *
 * Created on 23 August 2020, 17:28
 */

#ifndef IGRAPH_H
#define IGRAPH_H
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

#include "list/DLinkedList.h"


class VertexNotFoundException: public std::exception{
private:
    string vertex;
public:
    VertexNotFoundException(string vertex){
        this->vertex = vertex;
    }
    const char * what () const throw (){
        stringstream os;
        os << "Vertex (" << this->vertex << "): is not found";
        return os.str().c_str();
    }
};

class EdgeNotFoundException: public std::exception{
private:
    string edge;
public:
    EdgeNotFoundException(string edge){
        this->edge = edge;
    }
    const char * what () const throw (){
        stringstream os;
        os << "Edge (" << edge << "): is not found";
        return os.str().c_str();
    }
};

template<class T>
struct Edge{
    T from, to;
    float weight;
    Edge(T from, T to, float weight=0){
        this->from = from;
        this->to = to;
        this->weight = weight;
    };
    Edge(const Edge& edge){
        this->from = edge.from;
        this->to = edge.to;
        this->weight = edge.weight;
    }
};
/*
 * IGraph: define APIs for a graph data structure
 *  >> T: type of vertices
```

```
 */
template<class T>
class IGraph{
public:
    virtual ~IGraph(){};

    /*
    add (T vertex):
    add a vertex to graph
    */
    virtual void add(T vertex)=0;
    virtual void remove(T vertex)=0;
    virtual bool contains(T vertex)=0;

    /*
    connect(T from, T to, float weight):
    connect 2 vertexes (from -> to) with weight
    */
    virtual void connect(T from, T to, float weight=0)=0;
    virtual void disconnect(T from, T to)=0;
    virtual float weight(T from, T to)=0;

    virtual DLinkedList<T> getOutwardEdges(T from)=0;
    virtual DLinkedList<T> getInwardEdges(T to)=0;

    virtual int size()=0;
    virtual bool empty()=0;
    virtual void clear()=0;

    /*
    inDegree(T vertex):
    find the in degree of the vertex
    */
    virtual int inDegree(T vertex)=0;

    /*
    outDegree(T vertex):
    find the out degree of the vertex
    */
    virtual int outDegree(T vertex)=0;

    virtual DLinkedList<T> vertices()=0;
    virtual bool connected(T from, T to)=0;

    virtual string toString()=0;
};

/*
 * Path: model a path on graphs
 *  >> a path = sequence of vertices,
 *      -> stored in: "path" (DLinkedList<T>)
 *      -> its cost: stored in "cost" (float)
 *
 */
template<class T>
class Path{
private:
    DLinkedList<T> path;
    float cost;
public:
    Path(){
        cost = 0;
    }
    DLinkedList<T>& getPath(){
        return this->path;
    }
    float getCost(){
        return cost;
    }
```

```
139         void setCost(float cost){
140             this->cost = cost;
141         }
142 
143         ////////////////////////////////////////
144         void add(T item){
145             this->path.add(item);
146         }
147         string toString(string (*item2str)(T&)=0){
148             stringstream os;
149             os << this->path.toString(item2str)
150                     << ", cost: " << this->cost;
151             return os.str();
152         }
153     };
154 
155     /*
156      * IFinder: the path finder, contains searching algorithms on graph
157      *
158      */
159     template<class T>
160     class IFinder{
161         virtual DLinkedList<Path<T>> dijkstra(IGraph<T>* pGraph, T start)=0;
162     };
163 
164     #endif /* IGRAPH_H */
165 
166 
```