## CSCI 230 PA 11 Submission

## Due Date: ##/##/2022 Late (date and time):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Name(s): Nero Li

Exercise 1 & Exercise 2 & Extra Credit -- need to submit source code and I/O  
 -- check if completely done ✔️ ; otherwise, discuss issues below

Source code below:

**Entry.h:**

#ifndef ENTRY\_H

#define ENTRY\_H

// Modified for CSCI 220 Fall 15

// Updated Fall 21

template <typename K, typename V>

class Entry { // a (key, value) pair

public: // public functions

typedef K Key; // key type

typedef V Value; // value type

Entry(const K& k = K(), const V& v = V()) // constructor

: \_key(k), \_value(v) { }

const K& key() const { return \_key; } // get key

const V& value() const { return \_value; } // get value

void setKey(const K& k) { \_key = k; } // set key

void setValue(const V& v) { \_value = v; } // set value

bool operator==(const Entry a)

{

return (\_key == a.key() && \_value == a.value());

}

private: // private data

K \_key; // key

V \_value; // value

};

#endif

**HeapPriorityQueue.h:**

#ifndef HPQ\_H

#define HPQ\_H

#include <list>

#include <vector>

template <typename E>

class VectorCompleteTree

{

private: // member data

std::vector<E> V; // tree contents

public: // publicly accessible types

typedef typename std::vector<E>::iterator Position; // a position in the tree

protected: // protected utility functions

Position pos(int i) // map an index to a position

{ return V.begin() + i; }

int idx(const Position& p) const // map a position to an index

{ return p - V.begin(); }

public:

VectorCompleteTree() : V(1) {} // constructor

int size() const { return V.size() - 1; }

Position left(const Position& p) { return pos(2\*idx(p)); }

Position right(const Position& p) { return pos(2\*idx(p) + 1); }

Position parent(const Position& p) { return pos(idx(p)/2); }

bool hasLeft(const Position& p) const { return 2\*idx(p) <= size(); }

bool hasRight(const Position& p) const { return 2\*idx(p) + 1 <= size(); }

bool isRoot(const Position& p) const { return idx(p) == 1; }

Position root() { return pos(1); }

Position last() { return pos(size()); }

void addLast(const E& e) { V.push\_back(e); }

void swap(const Position& p, const Position& q) { E e = \*q; \*q = \*p; \*p = e; }

// New function added for CSCI 230 PA10

Position removeLast()

{

Position p = V.end();

p--;

V.pop\_back();

return p;

}

void remove(E e)

{

Position p = V.begin();

for (auto i : V)

{

if (i == e)

{

V.erase(p);

return;

}

p++;

}

}

};

template <typename E, typename C>

class HeapPriorityQueue

{

public:

int size() const; // number of elements

bool empty() const; // is the queue empty?

E insert(const E& e); // insert element

const E& min(); // minimum element

E removeMin(); // remove minimum

// New function added for CSCI 230 PA10

void replace(const E& oldElem, const E& newElem)

{

T.remove(oldElem);

insert(newElem);

}

private:

VectorCompleteTree<E> T; // priority queue contents

C isLess; // less-than comparator

// shortcut for tree position

typedef typename VectorCompleteTree<E>::Position Position;

};

template <typename E, typename C> // number of elements

int HeapPriorityQueue<E,C>::size() const

{

return T.size();

}

template <typename E, typename C> // is the queue empty?

bool HeapPriorityQueue<E,C>::empty() const

{

return size() == 0;

}

template <typename E, typename C> // minimum element

const E& HeapPriorityQueue<E,C>::min()

{

return \*(T.root()); // return reference to root element

}

template <typename E, typename C> // insert element

E HeapPriorityQueue<E,C>::insert(const E& e)

{

T.addLast(e); // add e to heap

Position v = T.last(); // e's position

while (!T.isRoot(v)) // up-heap bubbling

{

Position u = T.parent(v);

if (!isLess(\*v, \*u)) break; // if v in order, we're done

T.swap(v, u); // ...else swap with parent

v = u;

}

return e;

}

template <typename E, typename C> // remove minimum

E HeapPriorityQueue<E,C>::removeMin()

{

Position p;

if (size() == 1) // only one node?

p = T.removeLast(); // ...remove it

else

{

Position u = T.root(); // root position

T.swap(u, T.last()); // swap last with root

p = T.removeLast(); // ...and remove last

while (T.hasLeft(u)) // down-heap bubbling

{

Position v = T.left(u);

if (T.hasRight(u) && isLess(\*(T.right(u)), \*v))

v = T.right(u); // v is u's smaller child

if (isLess(\*v, \*u)) // is u out of order?

{

T.swap(u, v); // ...then swap

u = v;

}

else break; // else we're done

}

}

return \*p;

}

#endif

**Decorator.h:**

#pragma once

#include <string>

#include <map>

using namespace std;

// Created by T. Vo for CSCI 230

// Based on C++ code fragment of Goodrich book

class Object { // generic object

public:

virtual int intValue() const; // throw(bad\_cast);

virtual string stringValue() const ; // throw(bad\_cast);

};

class String : public Object {

private:

string value;

public:

String(string v = "") : value(v) { }

string getValue() const

{

return value;

}

};

class Integer : public Object {

private:

int value;

public:

Integer(int v = 0) : value(v) { }

int getValue() const

{

return value;

}

};

int Object::intValue() const // throw(bad\_cast) { // cast to Integer

{

const Integer\* p = dynamic\_cast<const Integer\*>(this);

if (p == NULL) throw exception(); // ("Illegal attempt to cast to Integer");

return p->getValue();

}

string Object::stringValue() const { // throw(bad\_cast) { // cast to String

const String\* p = dynamic\_cast<const String\*>(this);

if (p == NULL) throw exception(); // ("Illegal attempt to cast to Srring");

return p->getValue();

}

class Decorator {

private: // member data

std::map<string, Object\*> map1; // the map

public:

Object \* get(const string& a) // get value of attribute

{

return map1[a];

}

void set(const string& a, Object\* d) // set value

{

map1[a] = d;

}

};

**Graph.h:**

#pragma once

#include <vector>

#include <list>

#include <string>

#include "Decorator.h"

using namespace std;

// Created by T. Vo for CSCI 230

// Based on Java version of Goodrich book w/o template

// string for vertex and int for edge

// Version 1.1

class Vertex : public Decorator // behaves like interface in Java

{

public:

virtual string getElement() = 0;

};

class Edge : public Decorator // behaves like interface in Java

{

public:

virtual int getElement() = 0;

};

class Graph

{

public:

/\* Returns the number of vertices of the graph \*/

virtual int numVertices() = 0;

/\* Returns the number of edges of the graph \*/

virtual int numEdges() = 0;

/\* Returns the vertices of the graph as an iterable collection \*/

virtual list<Vertex \*> getVertices() = 0;

/\* Returns the edges of the graph as an iterable collection \*/

virtual list<Edge \*> getEdges() = 0;

/\*

\* Returns the number of edges leaving vertex v.

\* Note that for an undirected graph, this is the same result

\* returned by inDegree

\* throws IllegalArgumentException if v is not a valid vertex?

\*/

virtual int outDegree(Vertex \*v) = 0; // throws IllegalArgumentException;

/\*\*

\* Returns the number of edges for which vertex v is the destination.

\* Note that for an undirected graph, this is the same result

\* returned by outDegree

\* throws IllegalArgumentException if v is not a valid vertex

\*/

virtual int inDegree(Vertex \*v) = 0; // throws IllegalArgumentException;

/\*

\* Returns an iterable collection of edges for which vertex v is the origin.

\* Note that for an undirected graph, this is the same result

\* returned by incomingEdges.

\* throws IllegalArgumentException if v is not a valid vertex

\*/

virtual vector<Edge \*> outgoingEdges(Vertex \*v) = 0; // throws IllegalArgumentException;

/\*

\* Returns an iterable collection of edges for which vertex v is the destination.

\* Note that for an undirected graph, this is the same result

\* returned by outgoingEdges.

\* throws IllegalArgumentException if v is not a valid vertex

\*/

virtual vector<Edge \*> incomingEdges(Vertex \*v) = 0; // throws IllegalArgumentException;

/\*\* Returns the edge from u to v, or null if they are not adjacent. \*/

virtual Edge \*getEdge(Vertex \*u, Vertex \*v) = 0; // throws IllegalArgumentException;

/\*

\* Returns the vertices of edge e as an array of length two.

\* If the graph is directed, the first vertex is the origin, and

\* the second is the destination. If the graph is undirected, the

\* order is arbitrary.

\*/

virtual vector<Vertex \*> endVertices(Edge \*e) = 0; // throws IllegalArgumentException;

/\* Returns the vertex that is opposite vertex v on edge e. \*/

virtual Vertex \*opposite(Vertex \*v, Edge \*e) = 0; // throws IllegalArgumentException;

/\* Inserts and returns a new vertex with the given element. \*/

virtual Vertex \*insertVertex(string element) = 0;

/\*

\* Inserts and returns a new edge between vertices u and v, storing given element.

\*

\* throws IllegalArgumentException if u or v are invalid vertices, or if an edge already exists between u and v.

\*/

virtual Edge \*insertEdge(Vertex \*u, Vertex \*v, int element) = 0; // throws IllegalArgumentException;

/\* Removes a vertex and all its incident edges from the graph. \*/

virtual void removeVertex(Vertex \*v) = 0; // throws IllegalArgumentException;

/\* Removes an edge from the graph. \*/

virtual void removeEdge(Edge \*e) = 0; // throws IllegalArgumentException;

virtual void print() = 0;

};

**AdjacencyListGraph.h:**

#pragma once

#include <iostream>

#include <list>

#include <vector>

#include <map>

#include "Graph.h"

using namespace std;

// Created by T. Vo for CSCI 230

// Based on Java version of Goodrich book w/o template

// and minimal exception handling

// Version 1.1

// Some operations are incomplete and there are provisions

// to change from map to a list/vector for adjacency list

class AdjacencyListGraph : public Graph

{

private:

bool isDirected;

list<Vertex \*> vertices;

list<Edge \*> edges;

/\* A vertex of an adjacency map graph representation. \*/

class InnerVertex : public Vertex

{

private:

string element;

Vertex \*pos;

vector<pair<Vertex \*, Edge \*>> \*outgoing;

vector<pair<Vertex \*, Edge \*>> \*incoming;

public :

/\* Constructs a new InnerVertex instance storing the given element. \*/

InnerVertex(string elem, bool graphIsDirected = false) {

element = elem;

outgoing = new vector<pair<Vertex \*, Edge \*>>();

if (graphIsDirected)

incoming = new vector<pair<Vertex \*, Edge \*>>();

else

incoming = outgoing; // if undirected, alias outgoing map

}

/\* Returns the element associated with the vertex. \*/

string getElement() { return element; }

/\* Stores the position of this vertex within the graph's vertex list. \*/

void setPosition(Vertex \*p) { pos = p; }

/\* Returns the position of this vertex within the graph's vertex list. \*/

Vertex \*getPosition() { return pos; }

/\* Returns reference to the underlying map of outgoing edges. \*/

vector<pair<Vertex \*, Edge \*>> \*getOutgoing() { return outgoing; }

/\* Returns reference to the underlying map of incoming edges. \*/

vector<pair<Vertex \*, Edge \*>> \*getIncoming() { return incoming; }

}; //------------ end of InnerVertex class ------------

//---------------- nested InnerEdge class ----------------

/\* An edge between two vertices. \*/

class InnerEdge : public Edge

{

private:

double element;

Edge \*pos;

vector<Vertex \*> endpoints;

public:

/\* Constructs InnerEdge instance from u to v, storing the given element. \*/

InnerEdge(Vertex \*u, Vertex \*v, double elem)

{

element = elem;

endpoints.push\_back(u);

endpoints.push\_back(v);

}

/\* Returns the element associated with the edge. \*/

double getElement() { return element; }

/\* Returns reference to the endpoint array. \*/

vector<Vertex \*> getEndpoints() { return endpoints; }

/\* Stores the position of this edge within the graph's vertex list. \*/

void setPosition(Edge \*p) { pos = p; }

/\* Returns the position of this edge within the graph's vertex list. \*/

Edge \*getPosition() { return pos; }

}; //------------ end of InnerEdge class ------------

public:

/\*

\* Constructs an empty graph.

\* The parameter determines whether this is an undirected or directed graph.

\*/

AdjacencyListGraph(bool directed = true)

{

isDirected = directed;

}

~AdjacencyListGraph()

{

}

/\* Returns the number of vertices of the graph \*/

int numVertices()

{

return static\_cast<int>(vertices.size());

}

/\* Returns the number of edges of the graph \*/

int numEdges()

{

return static\_cast<int>(edges.size());

}

/\* Returns the vertices of the graph as an iterable collection \*/

list<Vertex \*> getVertices()

{

return vertices;

}

/\* Returns the edges of the graph as an iterable collection \*/

list<Edge \*> getEdges()

{

return edges;

}

/\*

\* Returns the number of edges leaving vertex v.

\* Note that for an undirected graph, this is the same result

\* returned by inDegree

\* throws IllegalArgumentException if v is not a valid vertex?

\*/

int outDegree(Vertex \*v) // throws IllegalArgumentException;

{

InnerVertex \*vert = static\_cast<InnerVertex \*>(v);

return static\_cast<int>(vert->getOutgoing()->size());

}

/\*\*

\* Returns the number of edges for which vertex v is the destination.

\* Note that for an undirected graph, this is the same result

\* returned by outDegree

\* throws IllegalArgumentException if v is not a valid vertex

\*/

int inDegree(Vertex \*v) // throws IllegalArgumentException;

{

InnerVertex \*vert = static\_cast<InnerVertex \*>(v);

return static\_cast<int>(vert->getIncoming()->size());

}

/\*

\* Returns an iterable collection of edges for which vertex v is the origin.

\* Note that for an undirected graph, this is the same result

\* returned by incomingEdges.

\* throws IllegalArgumentException if v is not a valid vertex

\*/

vector<Edge \*> outgoingEdges(Vertex \*v) // throws IllegalArgumentException;

{

vector<Edge \*> temp;

vector<pair<Vertex \*, Edge \*>> \*mapPtr = static\_cast<InnerVertex \*>(v)->getOutgoing();

for (auto it = mapPtr->begin(); it != mapPtr->end(); ++it) {

temp.push\_back(it->second);

}

return temp;

}

/\*

\* Returns an iterable collection of edges for which vertex v is the destination.

\* Note that for an undirected graph, this is the same result

\* returned by outgoingEdges.

\* throws IllegalArgumentException if v is not a valid vertex

\*/

vector<Edge \*> incomingEdges(Vertex \*v) // throws IllegalArgumentException;

{

vector<Edge \*> temp;

vector<pair<Vertex \*, Edge \*>> \*mapPtr = static\_cast<InnerVertex \*>(v)->getIncoming();

for (auto it = mapPtr->begin(); it != mapPtr->end(); ++it) {

temp.push\_back(it->second);

}

return temp;

}

/\* Returns the edge from u to v, or null if they are not adjacent. \*/

Edge \*getEdge(Vertex \*u, Vertex \*v) // throws IllegalArgumentException;

{

Edge \*temp = nullptr;

vector<Edge \*> out = outgoingEdges(u);

for (auto i : out)

if (opposite(u, i)->getElement() == v->getElement())

temp = i;

return temp; // origin.getOutgoing().get(v); // will be null if no edge from u to v

}

/\*

\* Returns the vertices of edge e as an array of length two.

\* If the graph is directed, the first vertex is the origin, and

\* the second is the destination. If the graph is undirected, the

\* order is arbitrary.

\*/

vector<Vertex \*> endVertices(Edge \*e) // throws IllegalArgumentException;

{

vector<Vertex \*> endpoints = static\_cast<InnerEdge \*>(e)->getEndpoints();

return endpoints;

}

/\* Returns the vertex that is opposite vertex v on edge e. \*/

Vertex \*opposite(Vertex \*v, Edge \*e) // throws IllegalArgumentException;

{

vector<Vertex \*> endpoints = static\_cast<InnerEdge \*>(e)->getEndpoints();

if (endpoints[0] == v)

return endpoints[1];

else

return endpoints[0];

}

/\* Inserts and returns a new vertex with the given element. \*/

Vertex \*insertVertex(string element)

{

Vertex \*v = new InnerVertex(element, isDirected);

vertices.push\_back(v);

static\_cast<InnerVertex \*>(v)->setPosition(vertices.back());

return v;

}

/\*

\* Inserts and returns a new edge between vertices u and v, storing given element.

\*

\* throws IllegalArgumentException if u or v are invalid vertices, or if an edge already exists between u and v.

\*/

Edge \*insertEdge(Vertex \*u, Vertex \*v, double element) // throws IllegalArgumentException;

{

Edge \* e = new InnerEdge(u, v, element);

edges.push\_back(e);

static\_cast<InnerEdge \*>(e)->setPosition(edges.back());

InnerVertex \*origin = static\_cast<InnerVertex \*>(u);

InnerVertex \*dest = static\_cast<InnerVertex \*>(v);

(origin->getOutgoing())->push\_back(pair<Vertex\*, Edge\*>(v, e));

(dest->getIncoming())->push\_back(pair<Vertex\*, Edge\*>(u, e));

return e;

}

/\* Removes a vertex and all its incident edges from the graph. \*/

void removeVertex(Vertex \*v) // throws IllegalArgumentException;

{

//for (Edge<E> e : vert.getOutgoing().values())

// removeEdge(e);

//for (Edge<E> e : vert.getIncoming().values())

// removeEdge(e);

//// remove this vertex from the list of vertices

//vertices.remove(vert.getPosition());

}

/\* Removes an edge from the graph. \*/

void removeEdge(Edge \*e) // throws IllegalArgumentException;

{

// remove this edge from vertices' adjacencies

//InnerVertex<V>[] verts = (InnerVertex<V>[]) edge.getEndpoints();

//verts[0].getOutgoing().remove(verts[1]);

//verts[1].getIncoming().remove(verts[0]);

//// remove this edge from the list of edges

//edges.remove(edge.getPosition());

}

void print()

{

for (auto itr = vertices.begin(); itr != vertices.end(); itr++)

{

cout << "Vertex " << (\*itr)->getElement() << endl;

if (isDirected)

cout << " [outgoing]";

cout << " " << outDegree(\*itr) << " adjacencies:";

for (auto e : outgoingEdges(\*itr))

cout << "(" << opposite(\*itr, e)->getElement() << ", " << e->getElement() << ")" << " ";

cout << endl;

if (isDirected)

{

cout << " [incoming]";

cout << " " << inDegree(\*itr) << " adjacencies:";

for (auto e : incomingEdges(\*itr))

cout << "(" << opposite(\*itr, e)->getElement() << ", " << e->getElement() << ")" << " ";

cout << endl;

}

}

}

};

**exercise.cpp:**

/\* Program: PA\_11\_exercise

Author: Nero Li

Class: CSCI 230

Date: 05/24/2022

Description:

Modify exercise 1 to include additional features and you can

just submit exercise 2 since it includes all features of

exercise 1. Additional graph processing algorithms such as

shortest paths can be added to this class or another class such

as GraphAlgorithms.

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include <iostream>

#include <iomanip>

#include <fstream>

#include <map>

#include <stack>

#include "AdjacencyListGraph.h"

#include "HeapPriorityQueue.h"

#include "Entry.h"

using namespace std;

class Flights

{

private:

AdjacencyListGraph G;

map<string, Vertex \*> airport;

vector<string> dest;

vector<double> price;

enum Status {VISITED, UNEXPLORED, DISCOVERY, BACK};

class comp

{

public:

bool operator()(Entry<double, Vertex \*> a, Entry<double, Vertex \*> b)

{

return (a.key() < b.key());

}

};

class comp2

{

public:

bool operator()(Entry<int, Vertex \*> a, Entry<int, Vertex \*> b)

{

return (a.key() < b.key());

}

};

void dijkstraPrice(Vertex \*src, map<Vertex \*, Vertex \*> &prev, map<Vertex \*, double> &cloud)

{

map<Vertex \*, double> D;

HeapPriorityQueue<Entry<double, Vertex \*>, comp> pq;

map<Vertex \*, Entry<double, Vertex \*>> pqTokens;

for (Vertex \*v : G.getVertices())

{

if (v == src)

D.insert(pair<Vertex \*, double>(v, 0));

else

D.insert(pair<Vertex \*, double>(v, INT\_MAX));

pqTokens.insert(pair<Vertex \*, Entry<double, Vertex \*>>(v, pq.insert(Entry<double, Vertex \*>(D[v], v))));

}

while (!pq.empty())

{

Entry<double, Vertex \*> entry = pq.removeMin();

double key = entry.key();

Vertex \*u = entry.value();

cloud.insert(pair<Vertex \*, double>(u, key));

pqTokens.erase(u);

for (Edge \*e : G.outgoingEdges(u))

{

Vertex \*v = G.opposite(u, e);

if (cloud.find(v) == cloud.end())

{

int wgt = e->getElement();

if (D[u] + wgt < D[v])

{

D[v] = D[u] + wgt;

pq.replace(pqTokens[v], Entry<double, Vertex \*>(D[v], v));

prev[v] = u;

}

}

}

}

}

void cheapestFlight(Vertex \*src, Vertex \*dest)

{

map<Vertex \*, Vertex \*> prev;

map<Vertex \*, double> cloud;

dijkstraPrice(src, prev, cloud);

stack<Vertex \*> output;

stack<double> outPrice;

Vertex \*cur = dest;

while (cur != src)

{

output.push(cur);

outPrice.push(G.getEdge(prev[cur], cur)->getElement());

cur = prev[cur];

}

cout << "Path:\n";

cout << src->getElement();

while (!output.empty())

{

cout << " -- $" << outPrice.top() << " --> " << output.top()->getElement();

output.pop();

outPrice.pop();

}

cout << ", $" << cloud[dest] << endl;

}

void cheapestRoundTrip(Vertex \*src, Vertex \*dest)

{

map<Vertex \*, Vertex \*> prev\_src;

map<Vertex \*, double> cloud\_src;

dijkstraPrice(src, prev\_src, cloud\_src);

map<Vertex \*, Vertex \*> prev\_dest;

map<Vertex \*, double> cloud\_dest;

dijkstraPrice(dest, prev\_dest, cloud\_dest);

stack<Vertex \*> output;

stack<double> outPrice;

Vertex \*cur = src;

while (cur != dest)

{

output.push(cur);

outPrice.push(G.getEdge(prev\_dest[cur], cur)->getElement());

cur = prev\_dest[cur];

}

while (cur != src)

{

output.push(cur);

outPrice.push(G.getEdge(prev\_src[cur], cur)->getElement());

cur = prev\_src[cur];

}

cout << "Path:\n";

cout << src->getElement();

while (!output.empty())

{

cout << " -- $" << outPrice.top() << " --> " << output.top()->getElement();

output.pop();

outPrice.pop();

}

cout << ", $" << cloud\_src[dest] + cloud\_dest[src] << endl;

}

void DFS(Vertex \*v, map<Vertex \*, Status> &label)

{

if (label[v] != DISCOVERY)

{

cout << " --> ";

}

label[v] = VISITED;

cout << v->getElement();

for (auto e : G.outgoingEdges(v))

{

Vertex \*u = G.opposite(v, e);

if (label[u] == UNEXPLORED)

DFS(u, label);

}

}

void visitAll(Vertex \*v)

{

map<Vertex \*, Status> label;

for (auto i : G.getVertices())

label[i] = UNEXPLORED;

label[v] = DISCOVERY;

cout << "Path:\n";

DFS(v, label);

cout << endl;

}

void fewestStop(Vertex \*src, Vertex \*dest)

{

map<Vertex \*, int> D;

map<Vertex \*, int> cloud;

map<Vertex \*, Vertex \*> prev;

HeapPriorityQueue<Entry<int, Vertex \*>, comp2> pq;

map<Vertex \*, Entry<int, Vertex \*>> pqTokens;

for (Vertex \*v : G.getVertices())

{

if (v == src)

D.insert(pair<Vertex \*, int>(v, 0));

else

D.insert(pair<Vertex \*, int>(v, INT\_MAX));

pqTokens.insert(pair<Vertex \*, Entry<int, Vertex \*>>(v, pq.insert(Entry<int, Vertex \*>(D[v], v))));

}

while (!pq.empty())

{

Entry<int, Vertex \*> entry = pq.removeMin();

int key = entry.key();

Vertex \*u = entry.value();

cloud.insert(pair<Vertex \*, int>(u, key));

pqTokens.erase(u);

for (Edge \*e : G.outgoingEdges(u))

{

Vertex \*v = G.opposite(u, e);

if (cloud.find(v) == cloud.end())

{

int wgt = 1;

if (D[u] + wgt < D[v])

{

D[v] = D[u] + wgt;

pq.replace(pqTokens[v], Entry<int, Vertex \*>(D[v], v));

prev[v] = u;

}

}

}

}

cout << "Path:\t\t\t";

stack<Vertex \*> output;

Vertex \*cur = dest;

while (cur != src)

{

output.push(cur);

cur = prev[cur];

}

cout << src->getElement();

while (!output.empty())

{

cout << " -> " << output.top()->getElement();

output.pop();

}

cout << endl;

cout << "Stops:\t\t\t" << cloud[dest] - 1 << endl;

}

public:

Flights(string str)

{

ifstream fin;

fin.open(str, ios::binary);

if (!fin)

return;

while (!fin.eof())

{

string cur;

int p{2};

double n;

while (p--)

{

fin >> cur;

if (!cur.empty())

{

dest.push\_back(cur);

if (airport.find(cur) == airport.end())

airport.insert(pair<string, Vertex \*>(cur, G.insertVertex(cur)));

}

}

fin >> n;

price.push\_back(n);

}

for (int i = 0, j = 0; i < dest.size(); i += 2, ++j)

G.insertEdge(airport[dest[i]], airport[dest[i + 1]], price[j]);

}

void controlPanel()

{

bool quit{false};

char choice{'Q'};

cout << fixed << setprecision(2);

while (!quit)

{

cout << "--------------------------------------------------------------------------\n";

cout << "0. Display all flights\n";

cout << "1. Find a cheapest flight from one airport to another airport\n";

cout << "2. Find a cheapest roundtrip from one airport to another airport\n";

cout << "3. Find an order to visit all airports starting from an airport\n";

cout << "4. Find a flight with fewest stops from one airport to another airport\n";

cout << "Q. Exit\n";

cout << "--------------------------------------------------------------------------\n";

cout << "Your choice: ";

cin >> choice;

cout << "--------------------------------------------------------------------------\n";

string first;

string second;

switch(choice)

{

case '0':

G.print();

break;

case '1':

cout << "Start from:\t\t";

cin >> first;

cout << "Go to:\t\t\t";

cin >> second;

cheapestFlight(airport[first], airport[second]);

break;

case '2':

cout << "Start from:\t\t";

cin >> first;

cout << "Go to:\t\t\t";

cin >> second;

cheapestRoundTrip(airport[first], airport[second]);

break;

case '3':

cout << "Start from:\t\t";

cin >> first;

visitAll(airport[first]);

break;

case '4':

cout << "Start from:\t\t";

cin >> first;

cout << "Go to:\t\t\t";

cin >> second;

fewestStop(airport[first], airport[second]);

break;

default:

quit = true;

}

}

}

};

int main()

{

Flights test("PA11Flights.txt");

test.controlPanel();

cout << "Author: Nero Li\n";

return 0;

}

Input/output below:

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 0

--------------------------------------------------------------------------

Vertex LAX

[outgoing] 2 adjacencies:(SEA, 199.99) (DFW, 189.00)

[incoming] 3 adjacencies:(SFO, 79.00) (DFW, 199.00) (MSY, 190.00)

Vertex SEA

[outgoing] 1 adjacencies:(ORD, 179.50)

[incoming] 1 adjacencies:(LAX, 199.99)

Vertex DFW

[outgoing] 2 adjacencies:(LAX, 199.00) (SFO, 99.99)

[incoming] 3 adjacencies:(LAX, 189.00) (ORD, 50.00) (MSY, 109.00)

Vertex SFO

[outgoing] 1 adjacencies:(LAX, 79.00)

[incoming] 1 adjacencies:(DFW, 99.99)

Vertex ORD

[outgoing] 2 adjacencies:(DFW, 50.00) (BOS, 179.00)

[incoming] 3 adjacencies:(BOS, 149.00) (JFK, 99.00) (SEA, 179.50)

Vertex BOS

[outgoing] 2 adjacencies:(ORD, 149.00) (JFK, 99.00)

[incoming] 1 adjacencies:(ORD, 179.00)

Vertex JFK

[outgoing] 3 adjacencies:(ORD, 99.00) (MIA, 49.00) (MSY, 220.00)

[incoming] 1 adjacencies:(BOS, 99.00)

Vertex MIA

[outgoing] 1 adjacencies:(MSY, 50.00)

[incoming] 1 adjacencies:(JFK, 49.00)

Vertex MSY

[outgoing] 2 adjacencies:(LAX, 190.00) (DFW, 109.00)

[incoming] 2 adjacencies:(JFK, 220.00) (MIA, 50.00)

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 1

--------------------------------------------------------------------------

Start from: LAX

Go to: JFK

Path:

LAX -- $199.99 --> SEA -- $179.50 --> ORD -- $179.00 --> BOS -- $99.00 --> JFK, $656.00

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 1

--------------------------------------------------------------------------

Start from: JFK

Go to: LAX

Path:

JFK -- $49.00 --> MIA -- $50.00 --> MSY -- $190.00 --> LAX, $289.00

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 2

--------------------------------------------------------------------------

Start from: LAX

Go to: JFK

Path:

LAX -- $199.99 --> SEA -- $179.50 --> ORD -- $179.00 --> BOS -- $99.00 --> JFK -- $49.00 --> MIA -- $50.00 --> MSY -- $190.00 --> LAX, $945.00

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 2

--------------------------------------------------------------------------

Start from: SEA

Go to: SFO

Path:

SEA -- $179.50 --> ORD -- $50.00 --> DFW -- $99.99 --> SFO -- $79.00 --> LAX -- $199.99 --> SEA, $606.00

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 3

--------------------------------------------------------------------------

Start from: LAX

Path:

LAX --> SEA --> ORD --> DFW --> SFO --> BOS --> JFK --> MIA --> MSY

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 4

--------------------------------------------------------------------------

Start from: JFK

Go to: LAX

Path: JFK -> MSY -> LAX

Stops: 1

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: 4

--------------------------------------------------------------------------

Start from: SFO

Go to: SEA

Path: SFO -> LAX -> SEA

Stops: 1

--------------------------------------------------------------------------

0. Display all flights

1. Find a cheapest flight from one airport to another airport

2. Find a cheapest roundtrip from one airport to another airport

3. Find an order to visit all airports starting from an airport

4. Find a flight with fewest stops from one airport to another airport

Q. Exit

--------------------------------------------------------------------------

Your choice: Q

--------------------------------------------------------------------------

Author: Nero Li

Answer for Question 1:

Ans

Answer for Question 2:

Ans