Jessica Sullivan

ID: 1282151

Class: COMSC-210-5067

Professor: Pentcheva

Source Code:

CS210\_Assignment10\_Graphs.cpp:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// main.cpp*

*// CS210\_Assignment9\_Heaps*

*//*

*// Created by Jessie Sully on 4/17/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#include <assert.h>

#include <iostream>

#include <sstream>

**using** **namespace** std;

*//#define RUN\_TESTS*

*//#define RUN\_INPUT\_TEST*

#include "Graph.h"

#include "DynamicArray.h"

**static** **const** string TEST\_INPUT = "input.txt";

**static** **const** string GRAPH\_INPUT1 = "MinPath1.txt";

**static** **const** string GRAPH\_INPUT2 = "MinPath2.txt";

#if defined(WIN32) || defined(\_WIN32)

#define PATH\_SEPARATOR "\\"

#else

#define PATH\_SEPARATOR "/"

#endif */\* RUN\_INPUT\_TEST1 \*/*

*// returns true if memory leak*

**bool** testMemoryLeak();

*// runs tests for the project.*

**void** runTests(**const** string &codePath);

*// runs the UI for the project.*

**void** runProgram(**const** string &codePath);

**void** printMenu(Graph &graph);

**void** manageMenu(Graph &graph);

**int** main(**int** argc, **const** **char** \* argv[]) {

*// programmer's identification*

cout << "Programmer: Jessica Sullivan" << endl;

cout << "Programmer's ID: 1282151" << endl;

cout << "File: " << \_\_FILE\_\_ << endl;

string codePath(\_\_FILE\_\_);

size\_t filePos1 = codePath.rfind(PATH\_SEPARATOR);

codePath = codePath.erase(filePos1 + 1, string::npos);

#ifdef RUN\_INPUT\_TEST

*// Override cin with a test input file.*

string testInputFile = codePath + TEST\_INPUT;

ifstream in(testInputFile.c\_str());

cin.rdbuf(in.rdbuf()); *//redirect std::cin to in.txt!*

#endif */\* RUN\_INPUT\_TEST \*/*

#ifdef RUN\_TESTS

runTests(codePath);

#else

runProgram(codePath);

#endif */\* RUN\_TESTS \*/*

testMemoryLeak();

**return** 0;

}

*// run tests for the project*

**void** runTests(**const** string &codePath) {

*// testing DynamicArray*

DynamicArray<**int**> array1;

assert(!array1.validateIndex(0));

array1.addObject(10);

array1.addObject(5);

array1.addObject(7);

assert(array1.getSize() == 3);

assert(array1.validateIndex(0));

assert(array1.validateIndex(1));

assert(array1.validateIndex(2));

assert(!array1.validateIndex(3));

assert(!array1.validateIndex(-1));

assert(array1[0] == 10);

assert(array1[1] == 5);

assert(array1[2] == 7);

*//assert(array1.getCapacity() == 4);*

cout << array1 << endl << endl;

array1.clear();

assert(array1.getSize() == 0);

assert(!array1.validateIndex(1));

*// small tests for Graph*

Graph graph1;

graph1.addVertex("SF");

graph1.addVertex("LA");

graph1.addEdge(0, 1, 80);

assert(graph1.getEdge(0)->mWeight == 80);

assert(graph1.getEdge(0)->mVertexIdx1 == 0);

assert(graph1.getEdge(0)->mVertexIdx1 == 0);

assert(graph1.getVertex(0)->mEdgeIdxs[0] == 0);

assert(graph1.getVertex(1)->mEdgeIdxs[0] == 0);

cout << graph1 << endl;

graph1.addVertex("Chicago");

cout << graph1 << endl;

graph1.addEdge(0, 2, 200);

cout << graph1 << endl;

graph1.addEdge(1, 2, 230);

cout << graph1 << endl;

graph1.addVertex("NY");

*// file 1 tests for Graph*

Graph graph2(codePath + GRAPH\_INPUT1);

cout << graph2;

**for**(**int** i = 0; i < graph2.getNumVertices(); i++) {

**for** (**int** j = 0; j < graph2.getNumVertices(); j++) {

assert(graph2.getMinPath(i, j, cout) == graph2.getMinPath(j, i, cout));

}

}

*// file 2 tests for Graph*

Graph graph3(codePath + GRAPH\_INPUT2);

assert(graph3.getVertex(6)->mName == "ST LOUIS");

cout << graph3;

**for**(**int** i = 0; i < graph3.getNumVertices(); i++) {

**for** (**int** j = 0; j < graph3.getNumVertices(); j++) {

assert(graph3.getMinPath(i, j, cout) == graph3.getMinPath(j, i, cout));

}

}

}

*// returns true if memory leak*

**bool** testMemoryLeak() {

**if** (sNumVertexObjects != 0 || sNumEdgeObjects != 0) {

cerr << "Num leaked vertices: " << sNumVertexObjects << endl;

cerr << "Num leaked edges: " << sNumEdgeObjects << endl;

**return** **true**;

}

**return** **false**;

}

**void** printMenu(Graph &graph) {

**for** (**int** i = 0; i < graph.getNumVertices(); i++) {

cout << i << ": " << graph.getVertex(i)->mName << endl;

}

cout << "Please select two vertices separated by a space to find the minimum distance between two points (enter -1 to exit): ";

}

**void** manageMenu(Graph &graph) {

string input;

stringstream ss;

**int** index1 = -1, index2 = -1;

printMenu(graph);

getline(cin, input);

**while** (input != "-1") {

ss.str(input);

ss >> index1 >> index2;

**if** (graph.validateVertexIdx(index1) && graph.validateVertexIdx(index2)

&& !ss.fail()) {

cout << endl;

graph.getMinPath(index1, index2, cout);

}

**else** {

cerr << "Error with entry: '" << input << "'" << endl << endl;

}

ss.clear();

index1 = -1;

index2 = -1;

printMenu(graph);

getline(cin, input);

}

}

*// runs the UI for the project.*

**void** runProgram(**const** string &codePath) {

Graph graph1(codePath + GRAPH\_INPUT1);

Graph graph2(codePath + GRAPH\_INPUT2);

manageMenu(graph1);

manageMenu(graph2);

}

Graph.h

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// GRAPH.H*

*// CS210\_Assignment10\_Graphs*

*//*

*// Created by Jessie Sully on 4/22/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#ifndef GRAPH\_H

#define GRAPH\_H

#include <iostream>

#include <fstream>

#include <sstream>

#include <float.h>

**using** **namespace** std;

#include "DynamicArray.h"

*// Edge*

**class** Graph;

**static** **int** sNumEdgeObjects = 0;

**struct** Edge {

*// update sNumEdgeObjects for bookkeeping.*

Edge() { sNumEdgeObjects++; }

*// update sNumEdgeObjects for bookkeeping.*

Edge(**const** Edge &edge) : mWeight(edge.mWeight), mVertexIdx1(edge.mVertexIdx1), mVertexIdx2(edge.mVertexIdx2), mEdgeIdx(edge.mEdgeIdx), mGraph(edge.mGraph) { sNumEdgeObjects++; }

*// update sNumEdgeObjects for bookkeeping.*

Edge(**const** **double** weight, **int** vertexIdx1, **int** vertexIdx2, **int** edgeIdx, **const** Graph\* graph) : mWeight(weight), mVertexIdx1(vertexIdx1), mVertexIdx2(vertexIdx2), mEdgeIdx(edgeIdx), mGraph(graph) { sNumEdgeObjects++; }

*// update sNumEdgeObjects for bookkeeping.*

~Edge() { sNumEdgeObjects--; mGraph = **nullptr**;};

*// copies member variables*

**const** Edge& **operator**=(**const** Edge& edge);

*// makes streaming easier*

**friend** ostream& **operator**<<(ostream& ostr, **const** Edge \*edge);

**friend** ostream& **operator**<<(ostream& ostr, **const** Edge &edge);

**double** mWeight = 0;

**int** mVertexIdx1 = -1;

**int** mVertexIdx2 = -1;

**int** mEdgeIdx = -1;

**const** Graph\* mGraph = **nullptr**;

};

**const** Edge& Edge::**operator**=(**const** Edge &edge) {

mWeight = edge.mWeight;

mVertexIdx1 = edge.mVertexIdx1;

mVertexIdx2 = edge.mVertexIdx2;

mEdgeIdx = edge.mEdgeIdx;

mGraph = edge.mGraph;

**return** \***this**;

}

*// Vertex*

**static** **int** sNumVertexObjects = 0;

**struct** Vertex {

Vertex() { sNumVertexObjects++; }

Vertex(**const** string &name, **int** index, Graph\* graph) : mName(name), mIndex(index), mGraph(graph) { sNumVertexObjects++; }

~Vertex() { sNumVertexObjects--; mGraph = **nullptr**; }

*// copies member variables*

**const** Vertex& **operator**=(**const** Vertex& vertex);

**const** string& getName() **const** { **return** mName; }

**void** addEdgeIdx(**int** edgeIdx) { mEdgeIdxs.addObject(edgeIdx); }

**int** getEdgeIdx(**int** index) **const** { **return** mEdgeIdxs[index]; }

**int** getNumEdges() **const** { **return** mEdgeIdxs.getSize(); }

**const** DynamicArray<**int**>& getEdges() { **return** mEdgeIdxs; }

*// makes streaming easier*

**friend** ostream& **operator**<<(ostream& ostr, **const** Vertex \*vertex);

string mName;

**int** mIndex = 0;

DynamicArray<**int**> mEdgeIdxs;

Graph\* mGraph = **nullptr**;

};

**const** Vertex& Vertex::**operator**=(**const** Vertex &vertex) {

mName = vertex.mName;

mIndex = vertex.mIndex;

mEdgeIdxs = vertex.mEdgeIdxs;

mGraph = vertex.mGraph;

**return** \***this**;

}

**class** Graph {

**public**:

Graph() {}

Graph(**const** string &inputFile);

~Graph() {}

**const** Vertex\* getVertex(**int** index) **const** { **return** &mVertices[index]; }

**const** Edge\* getEdge(**int** index) **const** { **return** &mEdges[index]; }

**int** getNumVertices() **const** { **return** mVertices.getSize(); }

**int** getNumEdges() **const** { **return** mEdges.getSize(); }

**void** addEdge(**int** index1, **int** index2, **double** weight);

**void** addVertex(**const** string &name)

{ mVertices.addObject(Vertex(name, mVertices.getSize(), **this**)); }

**bool** validateVertexIdx(**int** index) { **return** index >= 0 && index < mVertices.getSize(); }

**double** getMinPath(**int** startIndex, **int** endIndex, ostream &ostr) **const**;

**friend** ostream& **operator**<<(ostream& ostr, **const** Graph &graph);

**private**:

DynamicArray<Vertex> mVertices;

DynamicArray<Edge> mEdges;

};

*// should add error handling?*

Graph::Graph(**const** string &inputFile) {

string input, name;

**int** index1 = 0, index2 = 0;

**double** weight = 0;

ifstream fopenInput(inputFile);

stringstream ss;

**if** (fopenInput) {

**do** {

getline(fopenInput, input);

**if** (input == "-1") **break**;

ss.str(input);

ss >> index1 >> name;

**while** (ss >> input) {

name += ' ';

name += input;

}

**if** (index1 == mVertices.getSize()) { *// enforcing no holes*

addVertex(name);

}

ss.clear();

name.clear();

} **while** (index1 != -1);

**do** {

getline(fopenInput, input);

**if** (input == "-1") **break**;

ss.str(input);

ss >> index1 >> index2 >> weight;

**if** (mVertices.validateIndex(index1) && mVertices.validateIndex(index2) && weight > 0)

addEdge(index1, index2, weight);

ss.clear();

} **while** (index1 != -1);

fopenInput.close();

}

}

**void** Graph::addEdge(**int** index1, **int** index2, **double** weight) {

**if** (mVertices.validateIndex(index1) && mVertices.validateIndex(index2)) {

mEdges.addObject(Edge(weight, index1, index2, mEdges.getSize(), **this**));

mVertices[index1].addEdgeIdx(mEdges.getSize() - 1);

mVertices[index2].addEdgeIdx(mEdges.getSize() - 1);

**return**;

}

**throw** out\_of\_range("invalid index");

}

**double** Graph::getMinPath(**int** startIndex, **int** endIndex, ostream &ostr) **const** {

**const** Vertex\* curVertex = getVertex(startIndex);

**const** Edge\* curEdge = **nullptr**;

**int** numVertices = mVertices.getSize();

**bool** settledVertices[numVertices];

**double** minPath[numVertices];

string minPathS[numVertices];

**int** otherVertexIdx = -1;

**int** curVertexIdx = startIndex;

**double** curPath = 0;

**double** minPathAvailable = DBL\_MAX;

ostr << "Shortest Path From " << getVertex(startIndex)->mName << " to " << getVertex(endIndex)->mName << ':' << endl;

**for** (**int** i = 0; i < numVertices; i++) {

settledVertices[i] = **false**;

minPath[i] = DBL\_MAX;

minPathS[i].empty();

}

minPath[startIndex] = 0;

settledVertices[startIndex] = **true**;

**while**(!settledVertices[endIndex]) {

curVertex = getVertex(curVertexIdx);

settledVertices[curVertexIdx] = **true**;

minPathAvailable = DBL\_MAX;

**for** (**int** i = 0; i < curVertex->getNumEdges(); i++) {

curEdge = getEdge(curVertex->getEdgeIdx(i));

otherVertexIdx = (curEdge->mVertexIdx1 == curVertexIdx ?

curEdge->mVertexIdx2 : curEdge->mVertexIdx1);

**if** (!settledVertices[otherVertexIdx]) {

curPath = (curVertexIdx == startIndex ? curEdge->mWeight : minPath[curVertexIdx] + curEdge->mWeight);

**if** (minPath[otherVertexIdx] >= curPath) {

stringstream ss;

minPath[otherVertexIdx] = curPath;

ss << minPathS[curVertexIdx];

ss << curEdge;

minPathS[otherVertexIdx] = ss.str();

ss.flush();

}

}

}

**for** (**int** i = 0; i < numVertices; i++) {

**if** (!settledVertices[i] && (minPathAvailable > minPath[i])) {

curVertexIdx = i;

minPathAvailable = minPath[i];

}

}

}

ostr << minPathS[endIndex];

ostr << "Total Distance: " << minPath[endIndex] << endl << endl;

**return** minPath[endIndex];

}

ostream& **operator**<<(ostream& ostr, **const** Edge &edge) {

ostr << edge.mWeight << ' ' << edge.mGraph->getVertex(edge.mVertexIdx1)->getName()

<< " <-> " << edge.mGraph->getVertex(edge.mVertexIdx2)->getName() << endl;

**return** ostr;

}

ostream& **operator**<<(ostream& ostr, **const** Edge \*edge) {

ostr << edge->mWeight << ' ' << edge->mGraph->getVertex(edge->mVertexIdx1)->getName()

<< " <-> " << edge->mGraph->getVertex(edge->mVertexIdx2)->getName() << endl;

**return** ostr;

}

ostream& **operator**<<(ostream& ostr, **const** Vertex &vertex) {

ostr << vertex.mName << endl;

**for** (**int** i = 0; i < vertex.mEdgeIdxs.getSize(); i++) {

ostr << vertex.mGraph->getEdge(vertex.mEdgeIdxs[i]) << endl;

}

**return** ostr;

}

ostream& **operator**<<(ostream& ostr, **const** Graph &graph) {

**for** (**int** i = 0; i < graph.mVertices.getSize(); i++) {

ostr << graph.mVertices[i] << endl;

}

**return** ostr;

}

#endif */\* GRAPH\_H \*/*

DynamicArray.h

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// DynamicArray.h*

*// CS210\_Assignment10\_Graphs*

*//*

*// Created by Jessie Sully on 4/23/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#ifndef DYNAMICARRAY\_H

#define DYNAMICARRAY\_H

#include <iostream>

**using** **namespace** std;

#ifdef RUN\_TESTS

**static** **const** **int** DEFAULT\_SIZE = 2;

#else

**static** **const** **int** DEFAULT\_SIZE = 10;

#endif */\* RUN\_TESTS \*/*

**struct** Edge;

*// Dynamic Array*

**template** <**typename** T>

**class** DynamicArray {

**public**:

DynamicArray() { mArray = **new** T[DEFAULT\_SIZE]; }

*// Runs clearTree to free allocated space*

~DynamicArray();

T& **operator**[](**int** index) **const**;

**const** DynamicArray<T>& **operator**=(**const** DynamicArray<T> &array);

T\* getArray() **const** { **return** mArray; }

**int** getSize() **const** { **return** mSize; }

**int** getCapacity() **const** { **return** mCapacity; }

**void** addObject(**const** T &object);

**bool** isEmpty() **const** { **return** mSize == 0; }

**void** clear();

**bool** validateIndex(**int** index) { **return** index >= 0 && index < mSize; }

**template** <**typename** U>

**friend** ostream& **operator**<<(ostream& ostr, **const** DynamicArray<T> &array);

**private**:

**int** mCapacity = DEFAULT\_SIZE;

**int** mSize = 0;

T\* mArray = **nullptr**;

**void** increaseCapacity();

**void** copyNodes(T\* newArray) **const**;

};

**template** <**typename** T>

DynamicArray<T>::~DynamicArray() {

**delete** [] mArray;

}

**template** <**typename** T>

T& DynamicArray<T>::**operator**[](**int** index) **const** {

**if** (index < mSize && index >= 0) **return** mArray[index];

**throw** out\_of\_range("invalid index");

}

**template** <**typename** T>

**const** DynamicArray<T>& DynamicArray<T>::**operator**=(**const** DynamicArray<T> &array) {

**delete** [] mArray;

mCapacity = array.mCapacity;

mArray = **new** T[mCapacity];

mSize = array.mSize;

**for** (**int** i = 0; i < mSize; i++) {

mArray[i] = array.mArray[i];

}

**return** \***this**;

}

**template** <**typename** T>

**void** DynamicArray<T>::clear() {

**delete** [] mArray;

mSize = 0;

mCapacity = DEFAULT\_SIZE;

mArray = **new** T[DEFAULT\_SIZE];

}

**template** <**typename** T>

**void** DynamicArray<T>::copyNodes(T\* newArray) **const** {

**for**(**int** i = 0; i < mSize; i++) {

newArray[i] = mArray[i];

}

}

**template** <**typename** T>

**void** DynamicArray<T>::increaseCapacity() {

mCapacity += DEFAULT\_SIZE;

T\* largerArray = **new** T[mCapacity];

copyNodes(largerArray);

**delete** [] mArray;

mArray = largerArray;

}

**template** <**typename** T>

**void** DynamicArray<T>::addObject(**const** T &object) {

**if** (mSize == mCapacity) {

increaseCapacity();

}

mArray[mSize] = object;

mSize++;

}

**template** <**typename** T>

ostream& **operator**<<(ostream& ostr, **const** DynamicArray<T> &array) {

**for** (**int** i = 0; i < array.getSize(); i++) {

ostr << array[i] << endl;

}

**return** ostr;

}

*/\**

*void PriorityQueue::swapNodes(int index1, int index2) {*

*Node storageNode;*

*storageNode = mMaxHeap[index1];*

*mMaxHeap[index1] = mMaxHeap[index2];*

*mMaxHeap[index2] = storageNode;*

*}*

*\*/*

#endif */\* DynamicArray\_h \*/*

Output:

A screenshot of text

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

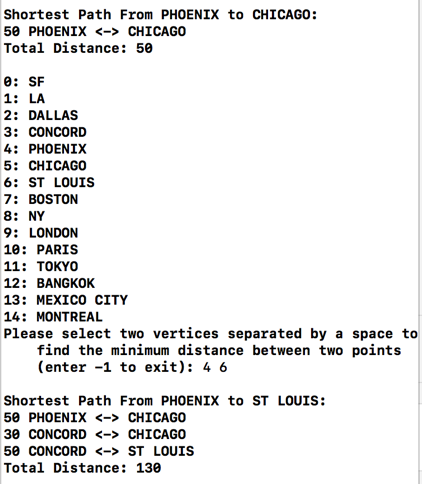
Description automatically generated

A close up of text on a white background

Description automatically generated

A screenshot of a cell phone

Description automatically generated

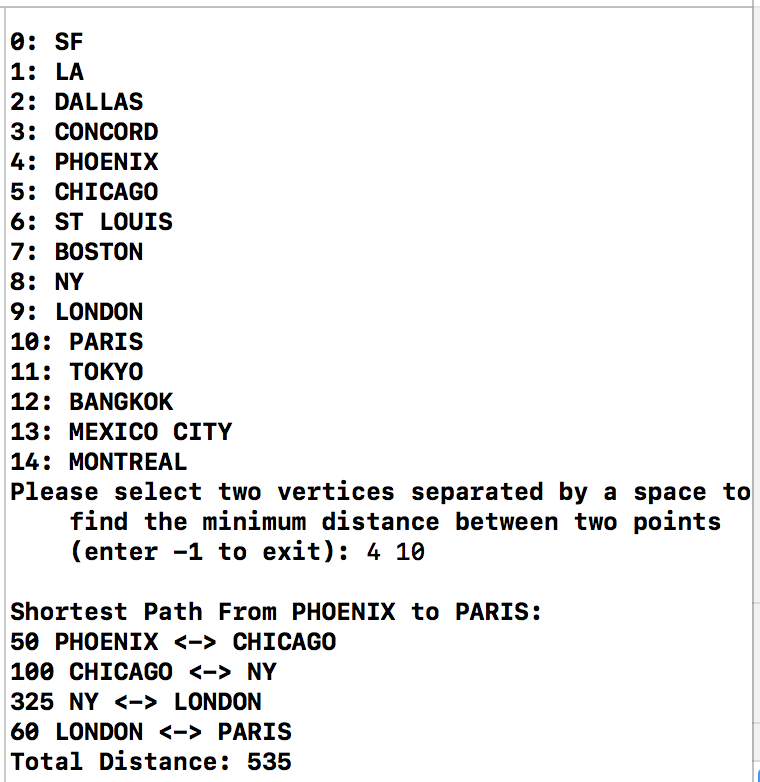


A screenshot of a cell phone

Description automatically generated

A close up of text on a white background

Description automatically generated



A screenshot of a cell phone

Description automatically generated

A close up of text on a white background

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of text on a white background

Description automatically generated

A picture containing bird

Description automatically generated