**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по курсовой работе**

**по дисциплине «Алгоритмы и Структуры Данных»**

**Вариант 3**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8301 |  | Чередилина Е. А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

Цель работы

Реализовать программу принимающую список ребер из файла, представляющий собой граф. Далее следует рассчитать максимальный поток в заданном графе методом «[Проталкивания предпотока](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BF%D1%80%D0%BE%D1%82%D0%B0%D0%BB%D0%BA%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B5%D0%B4%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0)».

Описание реализуемых и вспомогательных классов

Класс FlowCounterRepresenter, содержит поля: int\* redundantFlow (массив избытков вершин), int\*\* edgesCapacity (остаточная сеть), int\* height (функция высоты), int vertexCounter (количество вершин), int vertexFrom (исток), int vertexTo (сток).

Класс содержит следующие методы:

* Конструктор – получает на вход файл, в котором список строк, обрабатывает их и в результате выдает список смежности, где над диагональю пропускные способности прямых рёбер, а под диагональю пропускные способности «обратных рёбер».
* Деструктор – вызывает метод clear (на основе обычного удаления двоичного дерева).
* void pushFlow(int edge, int vertex) – функция, проталкивающая поток из u в v, равный min{e[edge],cf(edge, vertex)}, и пересчитывающая остаточную сеть и избытки.
* void edgeUpcoming(int edge) – функция, поднимающая вершину на минимальную высоту, достаточную для возможности проталкивания потока
* void pushingToLimit(int edge) – функция, выполняющая лифтинг и проталкивание, пока это возможно
* void checkStringCorrectness(int currentStringNumber, string fullCurrentLineString) – функция, проверяющая корректность текущей считанной из файла строки.
* int countMaxFlow() – функция, вычисляющая максимальный поток в сети

Оценка временной сложности алгоритмов

void pushFlow (int edge, int vertex) – O(1)

void edgeUpcoming (int edge) – O(|V|)

void pushingToLimit (int edge) – O(|V| |E|)

void checkStringCorrectness(int currentStringNumber, string fullCurrentLineString) – О(n), n – количество символов в строке.

int countMaxFlow () – O(|V|2 |E|).

Описание реализованных unit-тестов

Я реализовала unit-тесты для проверки ситуации c 6 вершинами и с 20.

В данном случае тесты проверяют ситуацию с одним ребром из стока в сток, а также и моменты, когда есть не только ребро из истока в сток, но и другие рёбра.

Тесты проверяют и корректность обработки исключительных ситуаций, например, когда пользователь не ввел одну из позиций, либо ввёл её некорректно (допустим поток с отрицательным знаком или не целое число), а также забыл ввести исток или сток.

**Обоснование выбора используемых структур данных**

При выполнении данной лабораторной работы я использовала МАР для того, чтобы индивидуализировать вершины индексами.

Данную структуру я выбрала потому, что она позволяет не сохранять повторяющиеся данные и быстрый доступ к ним.

List я использовала для перебора вершин сети в функции countMaxFlow(). В структуре List есть удобный функционал, что отличает ее от обычного массива - нам не нужно хранить размер массива. Также мы можем быстро добавлять и удалять элементы, без траты времени на их перезапись в новый массив (в нашем случае pushFront добавление в начало работает за O(1)).

Пример работы программы

|  |  |
| --- | --- |
| Входные данные | Вывод программы |
| S O 3  S P 3  O Q 3  O P 2  P R 2  Q R 4  Q T 2  R T 3 | Maximal flow value = 5 |

Листинг

|  |
| --- |
| FlowCounterRepresenter.h |
| #pragma once  #include <fstream>  #include "List.h"  #include<string>  #include"Map.h"  using namespace std;  template<typename T>  T min(T a, T b) {  return a > b ? b : a;  }  class FlowCounterRepresenter {  private:  int\* redundantFlow;  int\*\* edgesCapacity;  int\* height;  int vertexCounter;  int vertexFrom;  int vertexTo;    protected:  void pushFlow(int edge, int vertex)  {  int f = min(redundantFlow[edge], edgesCapacity[edge][vertex]);  redundantFlow[edge] -= f;  redundantFlow[vertex] += f;  edgesCapacity[edge][vertex] -= f;  edgesCapacity[vertex][edge] += f;  }  void edgeUpcoming(int edge)  {  int min = 2 \* vertexCounter + 1;  for (int i = 0; i < vertexCounter; i++)  if (edgesCapacity[edge][i] && (height[i] < min))  min = height[i];  height[edge] = min + 1;  }  void pushingToLimit(int edge)  {  int vertex = 0;  while (redundantFlow[edge] > 0)  {  if (edgesCapacity[edge][vertex] && height[edge] == height[vertex] + 1)  {  pushFlow(edge, vertex);  vertex = 0;  continue;  }  ++vertex;  if (vertex == vertexCounter)  {  edgeUpcoming(edge);  vertex = 0;  }  }  }  void checkStringCorrectness(int currentStringNumber, string fullCurrentLineString)  {  if (!((fullCurrentLineString[0] >= 'A' && fullCurrentLineString[0] <= 'Z') && (fullCurrentLineString[1] == ' '))) {  throw std::exception(string(("Error occuried while reading line " + to\_string(currentStringNumber))).c\_str());  }  if (!((fullCurrentLineString[2] >= 'A' && fullCurrentLineString[2] <= 'Z') && (fullCurrentLineString[3] == ' '))) {  throw std::exception(string(("Error occuried while reading line " + to\_string(currentStringNumber))).c\_str());  }  string currentLineWithNoSpaces;  for (int i = 4; i < fullCurrentLineString.size(); ++i) {  if (fullCurrentLineString[i] >= '0' && fullCurrentLineString[i] <= '9')  currentLineWithNoSpaces += fullCurrentLineString[i];  else {  throw std::exception(string(("Error occuried while reading line " + to\_string(currentStringNumber))).c\_str());  }  }  }  public:  int countMaxFlow() {  if (vertexCounter > 2) {  for (int i = 0; i < vertexCounter; i++)  {  if (i == vertexFrom)  continue;  redundantFlow[i] = edgesCapacity[vertexFrom][i];  edgesCapacity[i][vertexFrom] += edgesCapacity[vertexFrom][i];  }  height[vertexFrom] = vertexCounter;  List<int> temporaryList;  int current;  int currentIndex = 0;  int oldHeight;  for (int i = 0; i < vertexCounter; i++)  if (i != vertexFrom && i != vertexTo)  temporaryList.push\_front(i);  current = temporaryList.at(0);  while (currentIndex < temporaryList.get\_size())  {  oldHeight = height[current];  pushingToLimit(current);  if (height[current] != oldHeight)  {  temporaryList.push\_front(current);  temporaryList.remove(++currentIndex);  current = temporaryList.at(0);  currentIndex = 0;  }  ++currentIndex;  if (currentIndex < temporaryList.get\_size())  current = temporaryList.at(currentIndex);  }  if (redundantFlow[vertexTo] > 0)  return redundantFlow[vertexTo];  else  throw std::exception("Can't reach sink vertex form source.");  }  else  return edgesCapacity[0][1];  }  ~FlowCounterRepresenter() {  if(redundantFlow)  delete[] redundantFlow;    if(height)  delete[] height;    for(int i=0;i<vertexCounter;++i)  delete[] edgesCapacity[i];  }  FlowCounterRepresenter(ifstream& file)  {  Map<char, int>\* charToNumMap = new Map<char, int>();  vertexCounter = 0;  int currentStringNumber = 1;  while (!file.eof()) {  string fullCurrentLineString;  getline(file, fullCurrentLineString);  if (fullCurrentLineString.size() >= 5) {    checkStringCorrectness(currentStringNumber, fullCurrentLineString);  if (!charToNumMap->contains(fullCurrentLineString[0])) {  charToNumMap->insert(fullCurrentLineString[0], vertexCounter);  ++vertexCounter;  }  if (!charToNumMap->contains(fullCurrentLineString[2])) {  charToNumMap->insert(fullCurrentLineString[2], vertexCounter);  ++vertexCounter;  }  }  else  {  throw std::exception(string(("Error occuried while reading line " + to\_string(currentStringNumber))).c\_str());  }  ++currentStringNumber;  }  if (charToNumMap->contains('S'))  vertexFrom = charToNumMap->find('S');  else {  throw std::exception("No source vertex");  }  if (charToNumMap->contains('T'))  vertexTo = charToNumMap->find('T');  else {  throw std::exception("No sink vertex");  }  file.clear();  file.seekg(ios::beg);  redundantFlow = new int[vertexCounter];  height = new int[vertexCounter];  edgesCapacity = new int\* [vertexCounter];  for (int i = 0; i < vertexCounter; ++i) {  redundantFlow[i] = 0;  height[i] = 0;  }  for (int i = 0; i < vertexCounter; ++i) {  edgesCapacity[i] = new int[vertexCounter];  for (int j = 0; j < vertexCounter; ++j)  edgesCapacity[i][j] = 0;  }  currentStringNumber = 1;  while (!file.eof()) {  string s1;  int vert1, vert2, cap;  getline(file, s1);  vert1 = charToNumMap->find(s1[0]);  vert2 = charToNumMap->find(s1[2]);  if(vert1==vert2)  throw std::exception(string("Error occuried while reading line "+to\_string(currentStringNumber)).c\_str());  edgesCapacity[vert1][vert2] = stoi(s1.substr(4));  ++currentStringNumber;  }  }   1. }; |

|  |
| --- |
| UnitTestFlow\_Push\_Relabel.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include"..//CourseWork/FlowCounterRepresenter.h"  #include <fstream>  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UnitTestFlowPushRelabel  {  TEST\_CLASS(UnitTestFlowPushRelabel)  {  public:  TEST\_METHOD(TestMethod\_Сorrect\_output\_for\_6\_vertexes)  {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input1.txt");  FlowCounterRepresenter flow(input);  Assert::AreEqual(flow.countMaxFlow(), 5);  }  TEST\_METHOD(TestMethod\_Exception\_entering\_the\_first\_character) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input2.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(),"Error occuried while reading line 3");  }  }  TEST\_METHOD(TestMethod\_Exception\_entering\_the\_second\_character) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input3.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "Error occuried while reading line 3");  }  }  TEST\_METHOD(TestMethod\_Exception\_entering\_the\_third\_number\_flow) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input4.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "Error occuried while reading line 3");  }  }  TEST\_METHOD(TestMethod\_Exception\_empty\_string) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input5.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "Error occuried while reading line 3");  }  }  TEST\_METHOD(TestMethod\_Сorrect\_output\_for\_6\_vertexes\_and\_edge\_from\_source\_to\_sink)  {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input6.txt");  FlowCounterRepresenter flow(input);  Assert::AreEqual(flow.countMaxFlow(), 25);  }  TEST\_METHOD(TestMethod\_Сorrect\_output\_for\_2\_vertexes\_edges\_from\_source\_to\_sink)  {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input7.txt");  FlowCounterRepresenter flow(input);  Assert::AreEqual(flow.countMaxFlow(), 20);  }  TEST\_METHOD(TestMethod\_Exception\_there\_is\_a\_path\_from\_the\_vertex\_to\_itself) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input8.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "Error occuried while reading line 3");  }  }  TEST\_METHOD(TestMethod\_Сorrect\_output\_for\_20\_vertexes)  {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input9.txt");  FlowCounterRepresenter flow(input);  Assert::AreEqual(flow.countMaxFlow(), 19);  }  TEST\_METHOD(TestMethod\_Exception\_entering\_the\_third\_negative\_number\_flow) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input10.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "Error occuried while reading line 3");  }  }  TEST\_METHOD(TestMethod\_Exception\_sink\_is\_missing) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input11.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "No sink vertex");  }  }  TEST\_METHOD(TestMethod\_Exception\_source\_is\_missing) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input12.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "No source vertex");  }  }  TEST\_METHOD(TestMethod\_Exception\_two\_different\_connectivity\_components) {  try {  ifstream input("C:\\Users\\chere\\Desktop\\АиСД-4сем\\CourseWork\\CourseWork\\UnitTest\\Input13.txt");  FlowCounterRepresenter flow(input);  }  catch (exception & ex) {  Assert::AreEqual(ex.what(), "Can't reach sink vertex form source.");  }  }  };  } |

Результат работы тестов:

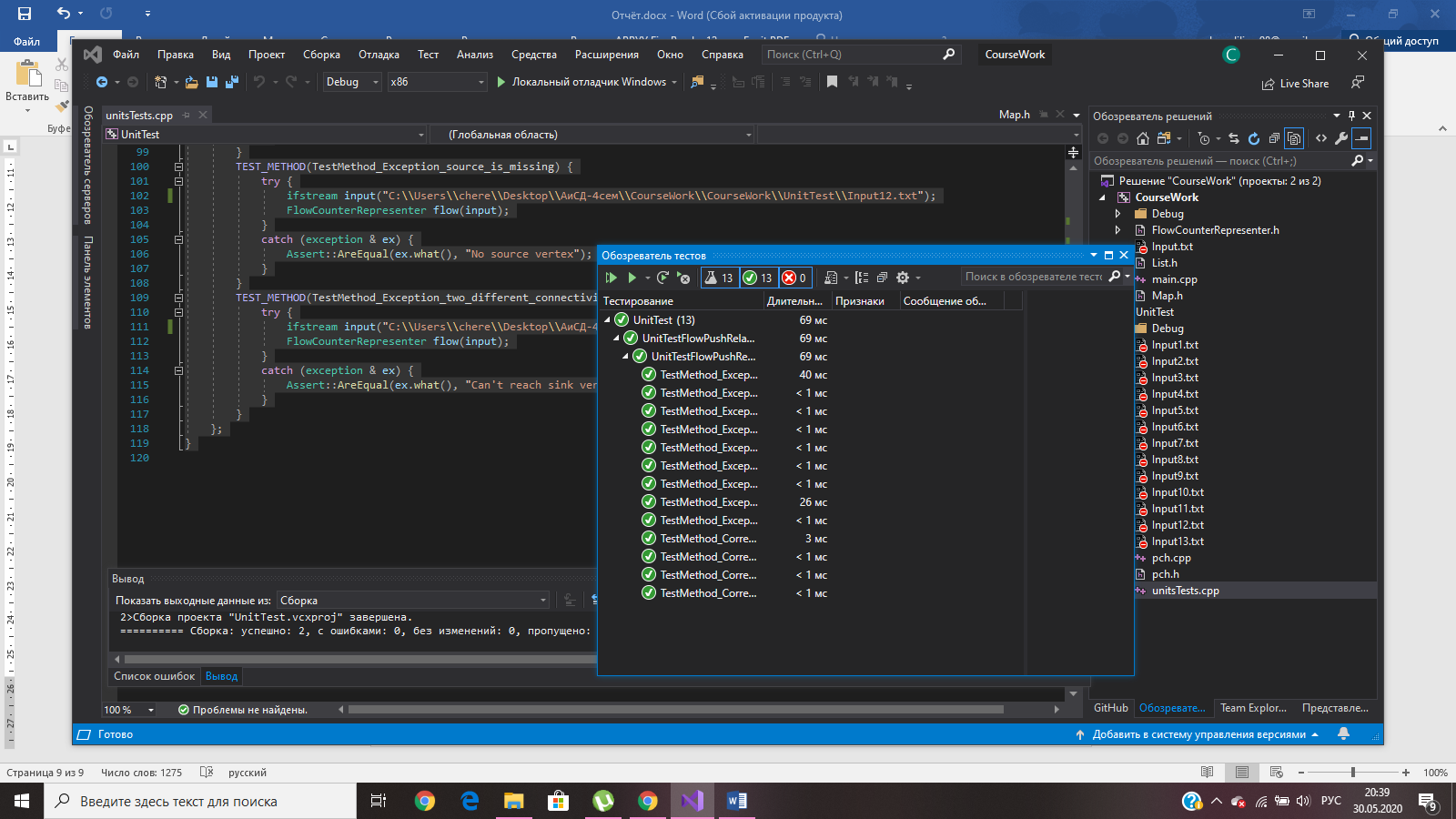


Рисунок 1 – Юнит тесты

Вывод

При выполнении данной лабораторной работы я ознакомилась и изучила работу алгоритма «Проталкивания предпотока», а также реализовала программу принимающую список ребер из файла, представляющий собой граф, и применила алгоритм для нахождения максимального потока в транспортной сети.