**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»**

**Тема: «Потоки в сетях»**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_ Тарабурин А.П.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2021

Оглавление

[1. Постановка задачи и описание реализуемого класса и методов 3](#_Toc55082721)

[2. Описание реализованных unit-тестов 4](#_Toc55082722)

[3. Код программы 5](#_Toc55082723)

[4. Пример работы 13](#_Toc55082724)

[5. Вывод 14](#_Toc55082725)

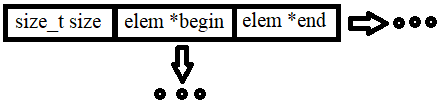
# Постановка задачи и описание реализуемого класса и методов

Найти максимальный поток в сети, используя алгоритм проталкивания предпотока

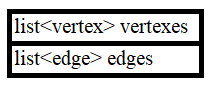
Class elem



Class list



Class graph



|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Оценка временной сложности** |
| void push\_back(int) | Добавление в конец списка | O(1) |
| void push\_front(int) | Добавление в начало списка | O(1) |
| void pop\_back() | Удаление последнего элемента | O(n) |
| void pop\_front() | Удаление первого элемента | O(1) |
| void insert(int, size\_t) | Добавление элемента по индексу | O(n) |
| T at(const int) | Получение элемента по индексу | O(n) |
| void remove(size\_t) | Удаление элемента по индексу | O(n) |
| size\_t getSize() | Получение размера списка | O(1) |
| void print\_to\_console() | Вывод элементов в консоль через разделитель | O(n) |
| void clear() | Удаление всех элементов списка | O(n) |
| void set(size\_t, int) | Замена элемента по индексу на передаваемый элемент | O(n) |
| bool isEmpty() | Проверка на пустоту списка | O(1) |
| void push\_front(list) | Вставка другого списка в начало | O(n) |
| T indexOf(int) | Получение индекса элемента в списке | O(n) |
| T back() | Получение значение последнего элемента | O(1) |
| **graph** | | |
| void inputVertex(fstream& in) | Ввод вершин из файла в список | O(n) |
| void inputEdge(fstream& in) | Ввод ребер из файла в список | O(n) |
| bool push(int uNumb) | Проталкивание избыточного потока | O(n2) |
| void relabel(int uNumb) | Проталкивание избыточного потока вверх | O(n) |
| void preflow(int indexStart) | Задать высоту и поток вершинам | O(n) |
| void updateReverseEdgeFlow(int indexArr, int flow) | Переназначение ребер | O(n) |
| int overFlowVertex(list<vertex>& temp) | Получение номера вершины с избыточным потоком | O(n) |
| int getMaxFlow() | Нахождение максимального потока в сети | O(V2\*E) |

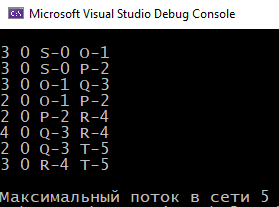
# Описание реализованных unit-тестов

|  |  |
| --- | --- |
| Имя теста | Описание |
| CreateTest | Проверка работы конструктора |
| PushBackTest1 | Проверка добавления элемента в конец пустого списка |
| PushBackTest2 | Проверка добавления элемента в конец непустого списка |
| PushFrontTest1 | Проверка добавления элемента в начало пустого списка |
| PushFrontTest2 | Проверка добавления элемента в начало непустого списка |
| PopBackTest1 | Проверка удаления элемента из конца непустого списка |
| PopBackTest2 | Проверка удаления элемента из конца списка с одним элементом |
| PopBackTest3 | Проверка удаления элемента из конца пустого списка |
| PopFrontTest1 | Проверка удаления элемента из начала непустого списка |
| PopFrontTest2 | Проверка удаления элемента из начала списка с одним элементом |
| PopFrontTest3 | Проверка удаления элемента из начала пустого списка |
| InsertTest1 | Проверка вставки элемента в пустой список на первую позицию |
| InsertTest2 | Проверка вставки элемента в пустой список на вторую позицию |
| InsertTest3 | Проверка вставки элемента в непустой список на посл. позицию |
| InsertTest4 | Проверка вставки элемента в непустой список на первую позицию |
| InsertTest5 | Проверка вставки элемента в середину непустого списка |
| AtTest1 | Проверка получения элемента по индексу |
| AtTest2 | Проверка получения элемента с первым индексом из пустого списка |
| RemoveTest1 | Проверка удаления элемента из пустого списка |
| RemoveTest2 | Проверка удаления элемента из непустого списка с посл. позиции |
| RemoveTest3 | Проверка удаления элемента из непустого списка с первой позиции |
| RemoveTest4 | Проверка удаления элемента из середины непустого списка |
| RemoveTest5 | Проверка удаления элемента с несуществующим индексом |
| GetSizeTest1 | Проверка получения размера пустого списка |
| GetSizeTest2 | Проверка получения размера непустого списка |
| SetTest1 | Проверка замены значения элемента непустого списка |
| SetTest2 | Проверка замены значения элемента с несуществующим индексом |
| SetTest3 | Проверка замены значения элемента непустого списка на такое же |
| SetTest4 | Проверка двойной замены элемента непустого списка |
| IsEmpty1 | Проверка определения пустоты списка |
| IsEmpty2 | Проверка определения пустоты списка |
| PushFrontListTest1 | Проверка вставки одного непустого в начало другого непустого |
| PushFrontListTest2 | Проверка вставки одного пустого в начало другого непустого |
| PushFrontListTest3 | Проверка вставки одного непустого в начало другого пустого |
| PushFrontListTest4 | Проверка вставки одного пустого в начало другого пустого |
| **graph** | |
| pushRelableSampleTest1 | Проверка на изначальных входных данных |
| pushRelableSampleTest2 | Проверка корректности работы алгоритма на иных входных данных |
| pushRelableSampleTest3 | Проверка корректности работы алгоритма на иных входных данных |
| pushRelableInputTest1 | Проверка корректности входных данных |

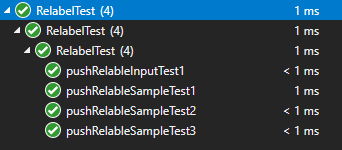
# Код программы

|  |
| --- |
| List.h |
| #pragma once  #include <iostream>  #include "graph.h"  #ifndef LIST\_H  #define LIST\_H  template<class T>  class elem {  private:  T inf;  elem\* next;  public:  elem(T elem\_inf) { inf = elem\_inf; next = nullptr;}  ~elem() {};  elem\* getNext() { return next; }  void setNext(elem\* elem\_next) { next = elem\_next; }  T getInf() { return inf; }  T\* getLink() { return &inf; }  void setInf(T elem\_inf) { inf = elem\_inf; }  };  template<class T>  class list {  private:  elem<T>\* begin, \* end;  size\_t size;  public:  list() { begin = nullptr; end = nullptr; size = 0;}  ~list() { clear(); }  //Adding new element to end of list  void push\_back(T temp) {  elem<T>\* newElem = new elem<T>(temp);  if (isEmpty()) begin = newElem;  else end->setNext(newElem);  end = newElem;  size++;  }  //Adding new element to begin of list  void push\_front(T temp) {  elem<T>\* newElem = new elem<T>(temp);  if (isEmpty()) end = newElem;  else newElem->setNext(begin);  begin = newElem;  size++;  }  //Deleting last element from list  void pop\_back(){  if (!isEmpty()) {  if (size == 1) {  end = nullptr;  begin = nullptr;  size = 0;  }  else {  elem<T>\* newEnd = begin;  while (newEnd->getNext() != end) newEnd = newEnd->getNext();  newEnd->setNext(nullptr);  elem<T>\* deleted = end;  delete deleted;  end = newEnd;  size--;  }  }  else throw "List is Empty";  }  //Deleting first element from list  void pop\_front() {  if (!isEmpty()) {  if (size == 1) {  begin = nullptr;  end = nullptr;  size = 0;  }  else {  elem<T>\* newBeg = begin->getNext();  elem<T>\* deleted = begin;  delete deleted;  begin = newBeg;  size--;  }  }  else throw "List is Empty";  }  //Adding element to any position in list  void insert(T data, size\_t pos) {  if (pos <= size) {  if (isEmpty() || pos == size) push\_back(data);  else {  if (pos == 0) push\_front(data);  else {  elem<T>\* newElem = new elem<T>(data);  elem<T>\* iter = begin;  while (pos-- > 1)  iter = iter->getNext();  newElem->setNext(iter->getNext());  iter->setNext(newElem);  size++;  }  }  }  else throw "Wrong index";  }  //Getting element from list by index  T\* at(size\_t pos) {  if (pos < size) {  elem<T>\* iter = begin;  while (pos-- != 0) iter = iter->getNext();  return iter->getLink();  }  else throw "Wrong index";  }  //Deleting element from list by index  void remove(size\_t pos) {  if (pos < size) {  if (pos == 0) pop\_front();  else {  if (pos == size - 1) pop\_back();  else {  elem<T>\* iter = begin;  while (pos-- > 1) iter = iter->getNext();  elem<T>\* nextElem = iter->getNext();  iter->setNext(nextElem->getNext());  size--;  }  }  }  else throw "Wrong index";  }  //Getting size of list  size\_t getSize() { return size; }  //Output elements from list to console  void print\_to\_console() {  elem<T>\* iter = begin;  for (size\_t i = 0; i < size; i++) {  std::cout << iter->getInf() << " ";  iter = iter->getNext();  }  }  //Deleting elements of list  void clear() {  while (size) pop\_back();  }  //Replacing element by index with new one  void set(size\_t pos, T data) {  if (pos < size) {  elem<T>\* iter = begin;  while (pos-- != 0) iter = iter->getNext();  iter->setInf(data);  }  else throw "Wrong index";  }  //Checking list for filling  bool isEmpty() {  if (size == 0) return true; // 1 - Empty  else return false; // 0 - Filled  }  //Adding another list to front of this one  void push\_front(list \*lst) {  for (size\_t i = 0; i < lst->getSize(); i++)  insert(lst->at(i), i);  }  //return last element in list  T back() {  return end->getInf();  }  //find element in list  bool contain(T toFind) {  elem<T>\* temp = begin;  while (temp != nullptr) {  if (temp->getInf() == toFind) return true;  temp = temp->getNext();  }  return false;  }  int indexOf(T toFind) {  elem<T>\* temp = begin;  int i = 0;  while (temp != nullptr) {  if (temp->getInf() == toFind) return i;  temp = temp->getNext();  i++;  }  return -1;  }  };  #endif |
| Main.cpp |
| #include <fstream>  #include <iostream>  #include "graph.h"  using namespace std;  int main() {  graph test;  try {  fstream in("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\in.txt");  test.inputVertex(in);  in.open("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\in.txt");  test.inputEdge(in);  }  catch (exception warning) {  cout << warning.what() << endl;  }  cout << endl;  try {  cout << "Максимальный поток в сети " << test.getMaxFlow();  }  catch (exception warning) {  cout << warning.what() << endl;  }  return 0;  } |
| RelabelTest.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "..\\pushRelabelMaxFlow\graph.h"  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace RelabelTest  {  TEST\_CLASS(RelabelTest)  {  public:  graph test;  TEST\_METHOD(pushRelableSampleTest1)  {  fstream in("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\in.txt");  test.inputVertex(in);  in.open("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\in.txt");  test.inputEdge(in);  Assert::AreEqual(test.getMaxFlow(), 5);  }  TEST\_METHOD(pushRelableSampleTest2)  {  fstream in("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\test1.txt");  test.inputVertex(in);  in.open("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\test1.txt");  test.inputEdge(in);  Assert::AreEqual(test.getMaxFlow(), 7);  }  TEST\_METHOD(pushRelableSampleTest3)  {  fstream in("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\test2.txt");  test.inputVertex(in);  in.open("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\test2.txt");  test.inputEdge(in);  Assert::AreEqual(test.getMaxFlow(), 9);  }  TEST\_METHOD(pushRelableInputTest1) {  fstream in("F:\\Универ\\Прога 4 сем\\AlgAndDataStruct\_4Sem\_TermWork\\pushRelabelMaxFlow\\incorrectTest.txt");  try {  test.inputVertex(in);  test.inputEdge(in);  }  catch (exception warning) {  Assert::AreEqual(warning.what(), "Ошибка во входных данных");  }  }  };  } |
| graph.h |
| #pragma once  #include "list.h"  #include <string>  #include <fstream>  using namespace std;  class vertex {  public:  char name = ' ';  int height = 0;  int edgeFlow = 0;  };  class edge {  public:  char uName = ' ';  int uNumb = 0;  char vName = ' ';  int vNumb = 0;  int flow = 0;  int capacity = -1;  };  class graph {  private:  list<edge> edges;  list<vertex> vertexes;  public:  graph() {  }    void inputVertex(fstream& in)  {  in >> noskipws;  if (!in.is\_open()) throw "Файл не открыт";  while (!in.eof()) {  char c = ' ';  vertex curVertex;  in >> c;  bool newVertex = true;  for (int i = 0; i < vertexes.getSize(); i++) {  if (vertexes.at(i)->name == c) {  newVertex = false;  break;  }  }  if (newVertex) {  curVertex.name = c;  vertexes.push\_back(curVertex);  }  in >> c;  if (c != ' ') throw "Ошибка во входных данных";  in >> c;  newVertex = true;  for (int i = 0; i < vertexes.getSize(); i++)  if (vertexes.at(i)->name == c) {  newVertex = false;  break;  }  if (newVertex) {  curVertex.name = c;  vertexes.push\_back(curVertex);  }  while (c != '\n' && !in.eof()) in >> c;  }  in.close();  }  void inputEdge(fstream& in)  {  if (vertexes.getSize() == 0) throw "Не заадны вершины";  in >> noskipws;  if (!in.is\_open()) throw "Файл не открыт";  char c;  while (!in.eof()) {  edge curEdge;  in.get(curEdge.uName);  in >> c;  if (c != ' ') throw "Ошибка во входных данных";  in >> curEdge.vName;  in >> c;  if (c != ' ') throw "Ошибка во входных данных";  in >> curEdge.capacity;  if (curEdge.capacity == -1) throw "Ошибка во входных данных";  for (int i = 0; i < vertexes.getSize(); i++) {  if (vertexes.at(i)->name == curEdge.uName) curEdge.uNumb = i;  if (vertexes.at(i)->name == curEdge.vName) curEdge.vNumb = i;  }  edges.push\_back(curEdge);  in << c;  }  cout << endl;  for (int i = 0; i < edges.getSize(); i++) {  cout << edges.at(i)->capacity << " ";  cout << edges.at(i)->flow << " ";  cout << edges.at(i)->uName << "-";  cout << edges.at(i)->uNumb << " ";  cout << edges.at(i)->vName << "-";  cout << edges.at(i)->vNumb << endl;  }  }  bool push(int uNumb)  {  for (int i = 0; i < edges.getSize(); i++) {  if (edges.at(i)->uNumb == uNumb) {  if (edges.at(i)->flow == edges.at(i)->capacity) continue;  if (vertexes.at(uNumb)->height > vertexes.at(edges.at(i)->vNumb)->height) {  int flow = min(edges.at(i)->capacity - edges.at(i)->flow,  vertexes.at(uNumb)->edgeFlow);  vertexes.at(uNumb)->edgeFlow -= flow;  vertexes.at(edges.at(i)->vNumb)->edgeFlow += flow;  edges.at(i)->flow += flow;  updateReverseEdgeFlow(i, flow);  return true;  }  }  }  return false;  }  void relabel(int uNumb)  {  int maxHeight = INT\_MAX;  for (int i = 0; i < edges.getSize(); i++){  if (edges.at(i)->uNumb == uNumb) {  if (edges.at(i)->flow == edges.at(i)->capacity) continue;  if (vertexes.at(edges.at(i)->vNumb)->height < maxHeight) {  maxHeight = vertexes.at(edges.at(i)->vNumb)->height;  vertexes.at(uNumb)->height = maxHeight + 1;  }  }  }  }  void preflow(int indexStart)  {  vertexes.at(indexStart)->height = vertexes.getSize();  for (int i = 0; i < edges.getSize(); i++) {  if (edges.at(i)->uNumb == indexStart) {  edges.at(i)->flow = edges.at(i)->capacity;  vertexes.at(edges.at(i)->vNumb)->edgeFlow += edges.at(i)->flow;  edge newEdge;  newEdge.flow = -edges.at(i)->flow;  newEdge.capacity = 0;  newEdge.uNumb = edges.at(i)->vNumb;  newEdge.uName = vertexes.at(edges.at(i)->vNumb)->name;  newEdge.vNumb = indexStart;  newEdge.vName = vertexes.at(indexStart)->name;  edges.push\_back(newEdge);  }  }  }  void updateReverseEdgeFlow(int indexArr, int flow)  {  int uNumb = edges.at(indexArr)->vNumb;  int vNumb = edges.at(indexArr)->uNumb;  for (int j = 0; j < edges.getSize(); j++) {  if (edges.at(j)->vNumb == vNumb && edges.at(j)->uNumb == uNumb) {  edges.at(j)->flow -= flow;  return;  }  }  edge newEdge;  newEdge.flow = 0;  newEdge.capacity = flow;  newEdge.uNumb = uNumb;  newEdge.uName = vertexes.at(uNumb)->name;  newEdge.vNumb = vNumb;  newEdge.vName = vertexes.at(vNumb)->name;  edges.push\_back(newEdge);  }  int overFlowVertex(list<vertex>& temp)  {  for (int i = 1; i < temp.getSize() - 1; i++)  if (temp.at(i)->edgeFlow > 0)  return i;  return -1;  }  int getMaxFlow() {  if (vertexes.getSize() == 0) throw "Не заданы вершины";  if (edges.getSize() == 0) throw "Не заданы ребра";  int indexStart = 0;  preflow(indexStart);  while (overFlowVertex(vertexes) != -1) {  int overFlowIndex = overFlowVertex(vertexes);  if (!push(overFlowIndex))  relabel(overFlowIndex);  }  return vertexes.back().edgeFlow;  }  }; |

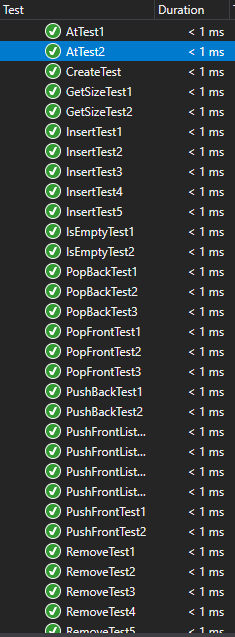
# Пример работы



RelabelTest



listUnitTest



# Вывод

При выполнении данной курсовой работы я познакомился с программной реализацией алгоритма проталкивания предпотока, с которой познакомился до этого на дисциплине Теория графов