**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: Алгоритм проталкивания предпотока.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8309 |  | Хваталов Д.И |
| Преподаватель |  | Тутуева А.В |

Санкт-Петербург

2020

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |  |
| --- | --- | --- | --- |
| **Студенту Хваталову Д.И.** | | | |
| **Группа 8309** | | | |
| **Тема работы: Алгоритмы на графах** | | | |
| **Исходные данные:**  Входные данные: текстовый файлы со строками в формате V1, V1, P, где V1, V2 направленная дуга транспортной сети, а P – ее пропускная способность. Исток всегда обозначен как S, сток – как T.  Пример файла для сети с изображения выше:  S O 3  S P 3  O Q 3  O P 2  P R 2  Q R 4  Q T 2  R T 3  Найти максимальный поток в сети используя алгоритм проталкивания предпотока. | | | | | |
| **Содержание пояснительной записки:**  «Исходная формулировка», «Цель работы», «Постановка задачи», «Обоснование выбора используемых структур данных», «Описание алгоритма решения», «Организация данных», «Пример работы», «Код программы». | | | |
| Дата выдачи задания: 15.03.2020 | | | |
| Студент |  | Хваталов Д.И. | |
| Преподаватель |  | Тутуева А.В | |

**Исходная формулировка задания:**

Входные данные: текстовый файлы со строками в формате V1, V1, P, где V1, V2 направленная дуга транспортной сети, а P – ее пропускная способность. Исток всегда обозначен как S, сток – как T

Пример файла для сети с изображения выше:

S O 3

S P 3

O Q 3

O P 2

P R 2

Q R 4

Q T 2

R T 3

Найти максимальный поток в сети используя алгоритм: Проталкивания предпотока.

**Цель работы:**

Создать программу, производящую поиск максимального потока в сети.

**Постановка задачи:**

Необходимо реализовать создание динамически расширяемого орграфа из файла, и найти в нем максимальный поток сети.

**Обоснование выбора используемых структур данных:**

Так как нам необходимо иметь быстрый доступ к информации мы будем использовать собственный контейнер “OURvec” основанный на динамическом массиве с реализацией вызова элементов через операторные скобки. Также создадим две структуры Edge и Vertex, первая будет хранить информацию о потоке и проходимости и название вершин, между которыми она расположена. Вторая структура будет хранить информацию о вершине, такую как проводимость, название и высоту. Сохранены все предметы в OURvec как уже было сказано для быстрого доступа элементам нашего графа.

**Описание алгоритма решения:**

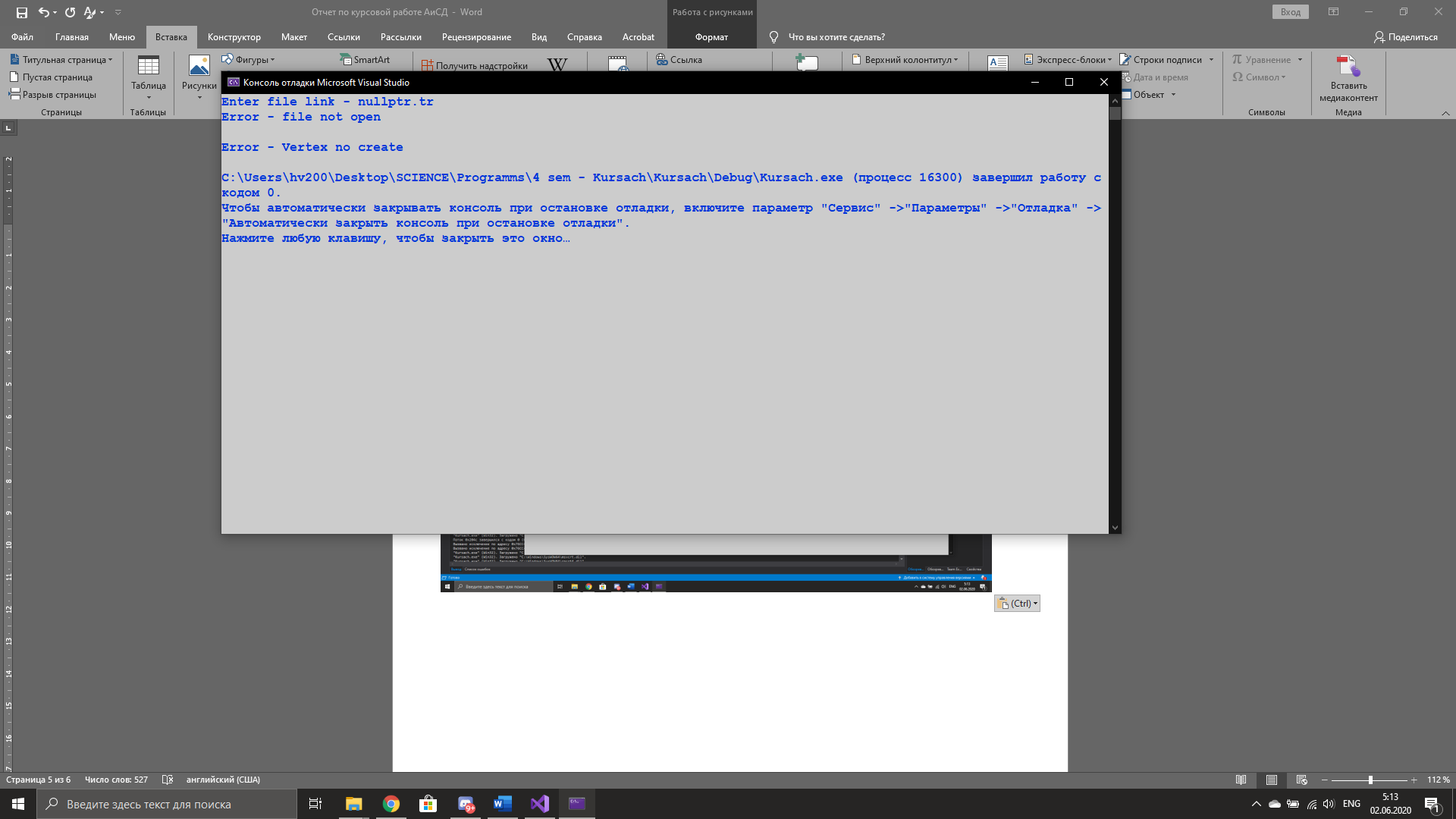
На каждом шаге будем рассматривать некоторый предпоток - т.е. функцию, которая по свойствам напоминает поток, но не обязательно удовлетворяет закону сохранения потока. На каждом шаге будем пытаться применить какую-либо из двух операций: проталкивание потока или поднятие вершины. Если на каком-то шаге станет невозможно применить какую-либо из двух операций, то мы нашли требуемый поток.

**Организация данных:**

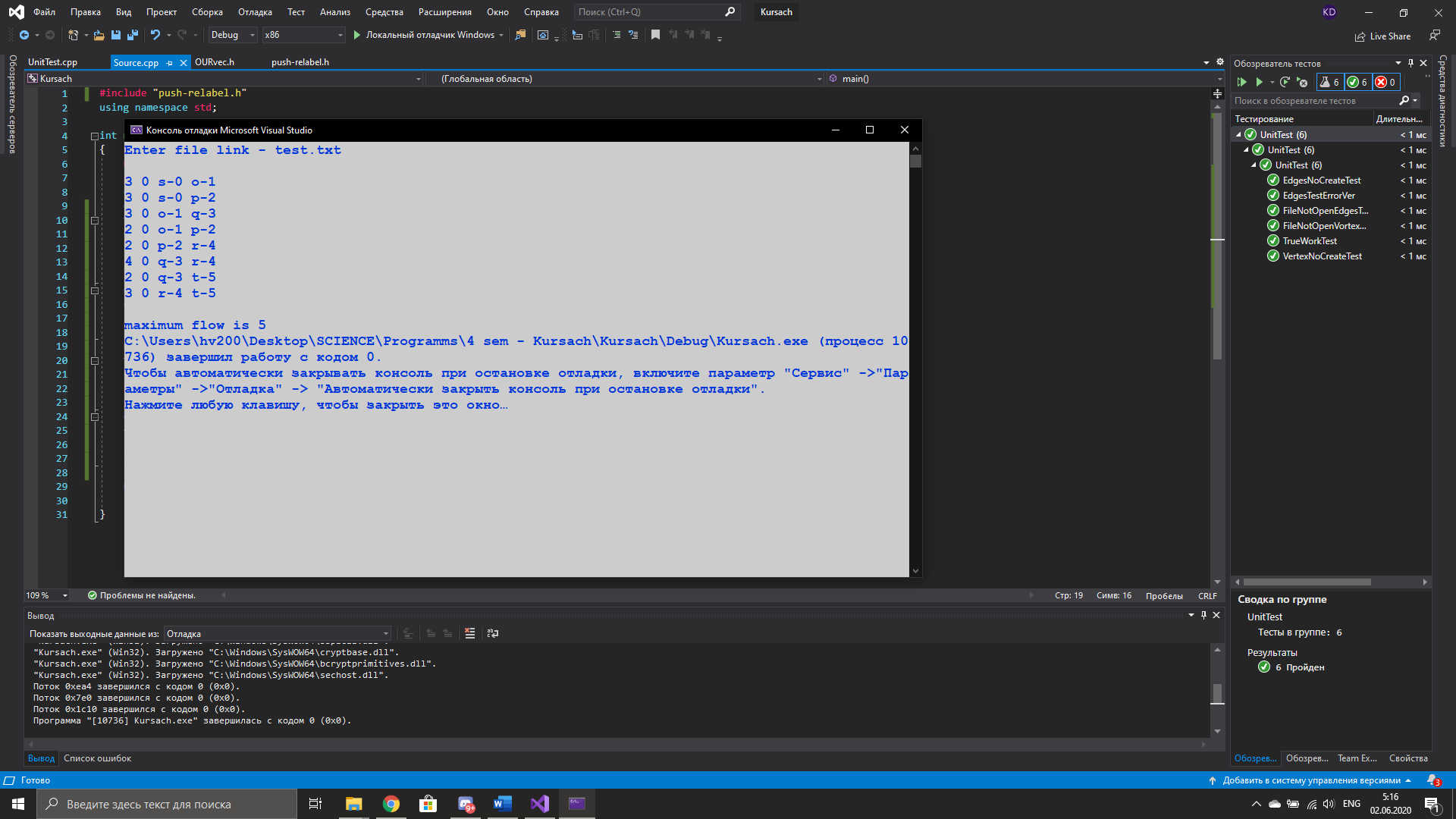
|  |  |  |
| --- | --- | --- |
| **Название** | **Описание работы метода** | **Оценка временной сложности** |
| int getMaxFlow(); | Возвращает максимальный поток в сети. | O(V^(2)\*E) |
| void addVertexs(string filelink) | Функция добавляет вершины в систему из текстового файла, путем прохода всего файла и поиска в системе уже существующих вершин. | O(N\*(K+K)) |
| void addEdge(string flink) | Функция добавляет ребра в систему из текстового файла и позицию вершины в OURvec. | O(N) |
| bool push(int u); | Используется для создания потока из узла, который имеет избыточный поток. Если у вершины есть избыточный поток, и есть соседний с меньшей высотой (в остаточном графе), мы продвигаем поток из вершины в соседний с меньшей высотой. Количество проталкиваемого потока через ребро равно минимуму избыточного потока и вместимости ребра. | O(N\*2) |
| void relabel(int u); | Используется, когда вершина имеет избыточный поток, и ни одна из ее смежных не находится на более низкой высоте. Мы в основном увеличиваем высоту вершины, чтобы мы могли выполнить push (). Чтобы увеличить высоту, мы выбираем минимальную смежную высоту и добавляем к ней 1. | O(N) |
| void preflow(int s); | Инициализирует высоты и потоки всех вершин | O(N) |
| void updateReverseEdgeFlow(int i, int flow); | Функция для изменения ребра в графе. | O(N) |
| int overFlowVertex(OURvector<Vertex>& ver) | Функция для возврата индекса переполненной вершины. | O(N) |

|  |  |
| --- | --- |
| **Название Unit-теста** | **Описание работы** |
| FileNotOpenVertexTest | Проверяем ошибку на некорректность файла при создании вершин. |
| EdgesTestErrorVer | Проверяем ошибку, при которой хотим создать ребра не объявив вершины. |
| FileNotOpenEdgesTest | Проверяем ошибку на некорректность файла при создании ребер. |
| VertexNoCreateTestMax | Проверка вызова максимального потока без созданных вершин. |
| EdgesNoCreateTestMax | Проверка вызова максимального потока без созданных ребер. |
| TrueWorkTest | Правильная работа поиска максимального потока в сети (ожидается 5). |

**Пример работы:**



**Рисунок 1 – На вход принят некорректный файл.**



**Рисунок 2 – пример правильной работы программы.**

**Код программы**

|  |
| --- |
| C:\Users\hv200\Desktop\SCIENCE\Programms\4 sem - Kursach\Kursach\push-relabel.h  #include<iostream>  #include<fstream>  #include<algorithm>  #include"OURvec.h"  class Graph  {  public:  Graph();  ~Graph();  int getMaxFlow();  void addVertexs(string filelink)  {  fstream file(filelink, ios::in);  if (!file.is\_open())  {  throw invalid\_argument("Error - file not open");  }  while (!file.eof())  {  char symb = ' ';  Vertex dt;  file.get(symb);  bool verNew = true;  for (int i = 0; i < ver.size(); i++)  {  if (ver[i].name == symb)  {  verNew = false;  break;  }  }  if (verNew == true)  {  dt.name = symb;  ver.push\_back(dt);  }  file.get(symb);  file.get(symb);  verNew = true;  for (int i = 0; i < ver.size(); i++)  {  if (ver[i].name == symb)  {  verNew = false;  break;  }  }  if (verNew == true)  {  dt.name = symb;  ver.push\_back(dt);  }  while (symb != '\n' && !file.eof())  {  file.get(symb);  }  }  file.close();  }  void addEdge(string flink)  {  if (ver.size() == 0)  {  throw out\_of\_range("Error - Vertex no creates");  }  fstream file(flink, ios::in);  if (!file.is\_open())  {  throw invalid\_argument("Error - file not open");  }  while (!file.eof())  {  Edge dt;  file.get(dt.uName);  file.get();  file.get(dt.vName);  file.get();  file >> dt.capacity;  for (int i = 0; i < ver.size(); i++)  {  if (ver[i].name == dt.uName)  {  dt.uNumb = i;  }  if (ver[i].name == dt.vName)  {  dt.vNumb = i;  }  }  edge.push\_back(dt);  file.get();  }  cout << endl;  for (int i = 0; i < edge.size(); i++)  {  cout<< edge[i].capacity<<" ";  cout << edge[i].flow << " ";  cout << edge[i].uName << "-";  cout << edge[i].uNumb << " ";  cout << edge[i].vName << "-";  cout << edge[i].vNumb <<endl;  }  }  struct Vertex  {  char name = ' ';  int height = 0;  int e\_flow = 0;  };  private:  struct Edge  {  int flow = 0;  int capacity = 0;  char uName = ' ';  int uNumb = 0;  char vName = ' ';  int vNumb = 0;  };  OURvector<Edge> edge;  OURvector<Vertex> ver;  bool push(int u);  void relabel(int u);  void preflow(int s);  void updateReverseEdgeFlow(int i, int flow);  int overFlowVertex(OURvector<Vertex>& ver)  {  for (int i = 1; i < ver.size() - 1; i++)  {  if (ver[i].e\_flow > 0)  return i;  }  return -1;  }  };  Graph::Graph()  {  }  Graph::~Graph()  {  }  void Graph::preflow(int s)  {  ver[s].height = ver.size();  for (int i = 0; i < edge.size(); i++)  {  if (edge[i].uNumb == s)  {  edge[i].flow = edge[i].capacity;  ver[edge[i].vNumb].e\_flow += edge[i].flow;  Edge ed;  ed.flow = -edge[i].flow;  ed.capacity = 0;  ed.uNumb = edge[i].vNumb;  ed.uName = ver[edge[i].vNumb].name;  ed.vNumb = s;  ed.vName = ver[s].name;  edge.push\_back(ed);  }  }  }  void Graph::updateReverseEdgeFlow(int i, int flow)  {  int u = edge[i].vNumb;  int v = edge[i].uNumb;  for (int j = 0; j < edge.size(); j++)  {  if (edge[j].vNumb == v && edge[j].uNumb == u)  {  edge[j].flow -= flow;  return;  }  }  Edge ed;  ed.flow = 0;  ed.capacity = flow;  ed.uNumb = u;  ed.uName = ver[u].name;  ed.vNumb = v;  ed.vName = ver[v].name;  edge.push\_back(ed);  }  bool Graph::push(int u)  {  for (int i = 0; i < edge.size(); i++)  {  if (edge[i].uNumb == u)  {  if (edge[i].flow == edge[i].capacity)  continue;  if (ver[u].height > ver[edge[i].vNumb].height)  {  int flow = min(edge[i].capacity - edge[i].flow,  ver[u].e\_flow);//Единственная строка за которую могу пояснить  ver[u].e\_flow -= flow;  ver[edge[i].vNumb].e\_flow += flow;  edge[i].flow += flow;  updateReverseEdgeFlow(i, flow);  return true;  }  }  }  return false;  }  void Graph::relabel(int u)  {  int mh = INT\_MAX;  for (int i = 0; i < edge.size(); i++)  {  if (edge[i].uNumb == u)  {  if (edge[i].flow == edge[i].capacity)  continue;  if (ver[edge[i].vNumb].height < mh)  {  mh = ver[edge[i].vNumb].height;  ver[u].height = mh + 1;  }  }  }  }  int Graph::getMaxFlow()  {  if (ver.size() == 0)  {  throw out\_of\_range("Error - Vertex no create");  }  if (edge.size() == 0)  {  throw out\_of\_range("Error - Edges no create");  }  int s = 0;  preflow(s);  while (overFlowVertex(ver) != -1)  {  int u = overFlowVertex(ver);  if (!push(u))  relabel(u);  }  return ver.back().e\_flow;  }  C:\Users\hv200\Desktop\SCIENCE\Programms\4 sem - Kursach\Kursach\Source.cpp  #include "push-relabel.h"  using namespace std;  int main()  {  Graph mg;  string flink;  cout << "Enter file link - ";  cin >> flink;  try  {  mg.addVertexs(flink);  mg.addEdge(flink);  }  catch (invalid\_argument error)  {  cout << error.what() << endl;  }  cout << endl;  try  {  cout << "maximum flow is " << mg.getMaxFlow();  }  catch (out\_of\_range error)  {  cout << error.what() << endl;  }  return 0;  } |