**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 1**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9301 |  | Кнауб К.В. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2021

# Постановка задачи

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: Форда — Фалкерсона. Данный алгоритм решает задачу нахождения максимального потока в транспортной сети. Затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника s к стоку t, вдоль которого можно послать больший поток). Процесс повторяется, пока можно найти увеличивающий путь.

# Описание реализуемых алгоритмов и используемых структур

Хранение графа осуществляется с помощью матрицы смежности. Обходом в глубину происходит поиск пути, после этого ребра графа меняются в зависимости от минимального потока на этом пути.

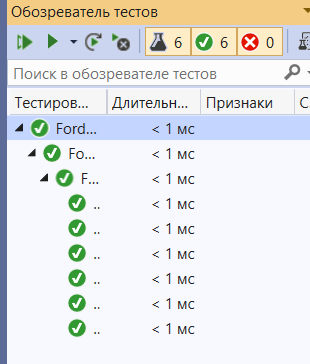
|  |  |
| --- | --- |
| **Название** | **Описание** |
| bool isContain(char\* arr, char ch) | определяет наличие символа ch в массиве символов arr |
| struct set | структура для предварительной обработки, которая хранит уникальные имена вершин и их количество |
| void FordFulkerson::inputData(string fileName) | считывает данные из входного файла и составляет матрицу смежности |
| int FordFulkerson::MaxStream() | поиск максимального потока |
| int FordFulkerson::dfs(int u, int Cmin, bool\* visited) | обход в глубину |
| int min(int a, int b) | возвращает минимальный элемент из двух подаваемых |

# Оценка временной сложности

|  |  |
| --- | --- |
| **Название метода** | **Оценка временной сложности** |
| inputData() | O(n) |
| MaxStream() | O(n2) |

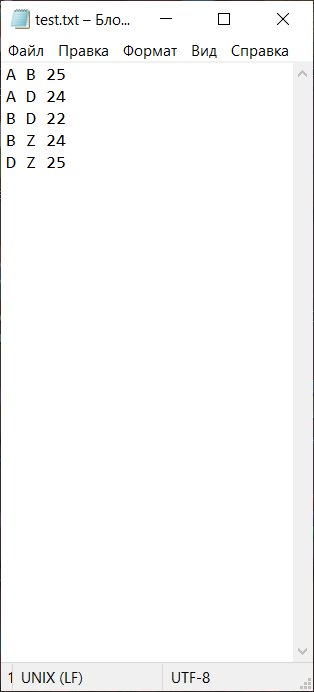
# Описание реализованных unit-тестов

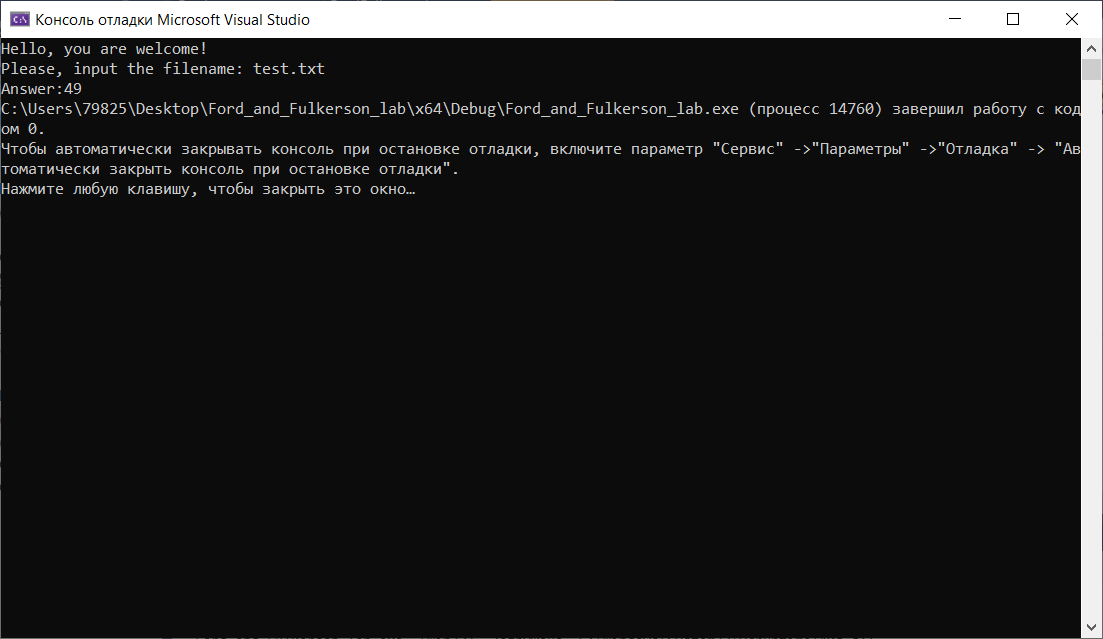
|  |
| --- |
| **Название** |
| Test1 |
| Test2 |
| Test3 |
| Test4 |
| Test5 |
| Test6 |



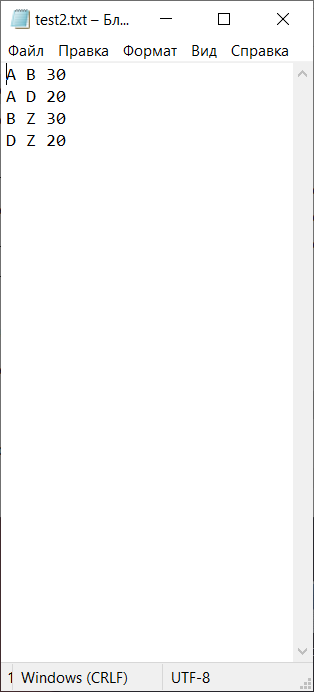
# Пример работы

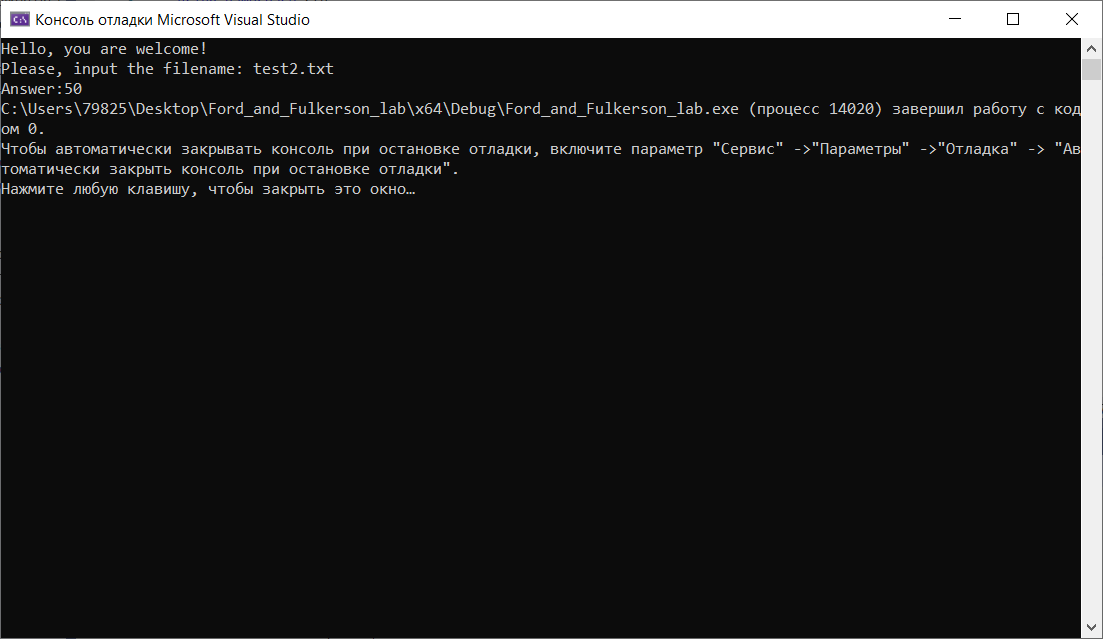
1 пример



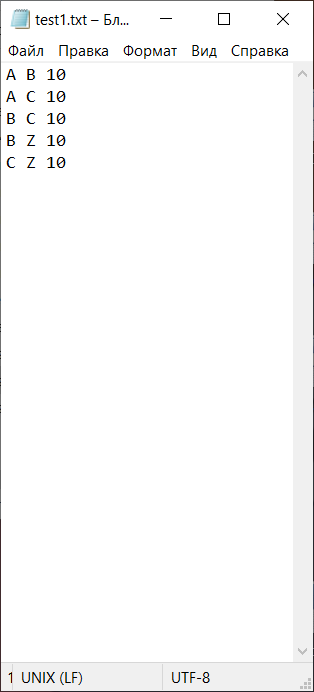


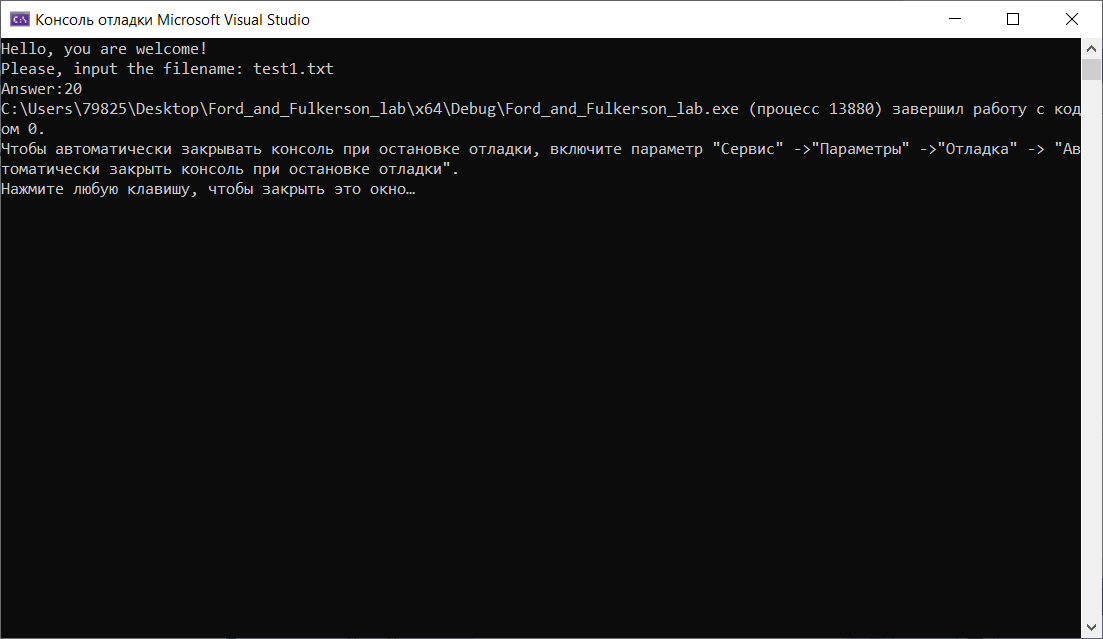
2 пример





3 пример





# Листинг

|  |
| --- |
| **main.cpp** |
| #include <iostream>  #include "FordFulkerson.h"  using namespace std;  int main()  {  cout << "Hello, you are welcome!\n";  cout << "Please, input the filename: ";  string filename;  //filename = "test.txt";  cin >> filename;  FordFulkerson alg;  alg.inputData(filename);  int res = alg.MaxStream();  cout <<"Answer:" << res;  } |
| **FordFulkerson.h** |
| #ifndef FORDFULKERSON\_H  #define FORDFULKERSON\_H  #define infinity INT\_MAX  #include <fstream>  #include <iostream>  #include <string>  using namespace std;  int min(int a, int b) {  if (a < b)  return a;  else  return b;  }  bool isContain(char\* arr, char ch) {  for (int i = 0; i < 26; i++) {  if (arr[i] == ch)  return true;  }  return false;  }  struct set {  char\* arr = new char[26];  int size = 0;  };  class FordFulkerson  {  public:  int Tops;  int\*\* graphMat;  int from, to;  int \_stream;  char\* TopsName;  int dfs(int u, int Cmin, bool\* visited) {  if (u == to)  return Cmin;  visited[u] = true;  int delta;  for (int v = 0; v < Tops; v++)  {  if (!visited[v] && (graphMat[u][v] > 0))  {  delta = dfs(v, min(Cmin, graphMat[u][v]), visited);  if (delta > 0)  {  graphMat[u][v] -= delta;  graphMat[v][u] += delta;  return delta;  }  }  }  return 0;  }  int MaxStream() {  \_stream = 0;  int toAdd = 0;  int buf1, buf2;  bool\* visited = new bool[Tops];  do  {  for (int i = 0; i < Tops; i++)  visited[i] = false;  toAdd = dfs(from, infinity, visited);  \_stream += toAdd;  } while (toAdd > 0);  return \_stream;  };  void inputData(string fileName) {  string temp;  fstream file;  char firstNode, secondNode;  int weight;  int edge = 0;  int CountTopName = 0;  file.open(fileName);  set \_set;  while (!file.eof()) {  temp = "";  getline(file, temp);  if (!isContain(\_set.arr, temp[0])) {  \_set.arr[\_set.size] = temp[0];  \_set.size++;  }  if (!isContain(\_set.arr, temp[2])) {  \_set.arr[\_set.size] = temp[2];  \_set.size++;  }  edge++;  }  Tops = \_set.size;  file.close();  file.open(fileName);  TopsName = new char[Tops];  int\*\* arr\_check = new int\* [Tops];  graphMat = arr\_check;  for (int i = 0; i < Tops; i++)  {  arr\_check[i] = new int[Tops];  graphMat[i] = arr\_check[i];  for (int j = 0; j < Tops; j++) {  graphMat[i][j] = 0;  }  }  int found1, found2;  for (int i = 0; i < edge; i++)  {  file >> firstNode >> secondNode >> weight;  found1 = -1;  for (int j = 0; j < CountTopName; j++)  if (TopsName[j] == firstNode)  found1 = j;  if (found1 == -1)  {  TopsName[CountTopName] = firstNode;  found1 = CountTopName;  CountTopName++;  }  found2 = -1;  for (int j = 0; j < CountTopName; j++)  if (TopsName[j] == secondNode)  found2 = j;  if (found2 == -1)  {  TopsName[CountTopName] = secondNode;  found2 = CountTopName;  CountTopName++;  }  graphMat[found1][found2] = weight;  }  file.close();  for (int i = 0; i < Tops; i++)  {  if (TopsName[i] == 'A')  from = i;  else if (TopsName[i] == 'Z')  to = i;  }  };  };  #endif // !FORDFULKERSON\_H |
| **Ford\_test.cpp** |
| #include "pch.h"  #include "CppUnitTest.h"  #include "..\Ford\_and\_Fulkerson\_lab\FordFulkerson.h"  #include "..\Ford\_and\_Fulkerson\_lab\main.cpp"  #define TEST\_CASE\_DIRECTORY GetDirectoryName(\_\_FILE\_\_)  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace Fordtest  {  TEST\_CLASS(Fordtest)  {  string GetDirectoryName(string path) {  const size\_t last\_slash\_idx = path.rfind('\\');  if (std::string::npos != last\_slash\_idx)  {  return path.substr(0, last\_slash\_idx + 1);  }  return "";  }  public:  TEST\_METHOD(test1)  {  string fileName = std::string(TEST\_CASE\_DIRECTORY) + "test1.txt";  FordFulkerson test;  test.inputData(fileName);  Assert::AreEqual(test.MaxStream(), 7);  }  TEST\_METHOD(test2)  {  string fileName = std::string(TEST\_CASE\_DIRECTORY) + "test2.txt";  FordFulkerson test;  test.inputData(fileName);  Assert::AreEqual(test.MaxStream(), 2000);  }  TEST\_METHOD(test3)  {  string fileName = std::string(TEST\_CASE\_DIRECTORY) + "test3.txt";  FordFulkerson test;  test.inputData(fileName);  Assert::AreEqual(test.MaxStream(), 30);  }  TEST\_METHOD(test4)  {  string fileName = std::string(TEST\_CASE\_DIRECTORY) + "test4.txt";  FordFulkerson test;  test.inputData(fileName);  Assert::AreEqual(test.MaxStream(), 10);  }  TEST\_METHOD(test5)  {  string fileName = std::string(TEST\_CASE\_DIRECTORY) + "test5.txt";  FordFulkerson test;  test.inputData(fileName);  Assert::AreEqual(test.MaxStream(), 10);  }  TEST\_METHOD(test6)  {  string fileName = std::string(TEST\_CASE\_DIRECTORY) + "test6.txt";  FordFulkerson test;  test.inputData(fileName);  Assert::AreEqual(test.MaxStream(), 10);  }  };  } |