МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

Курсовая работа
по дисциплине «Алгоритмы и структуры данных»
Тема: Потоки в сетях

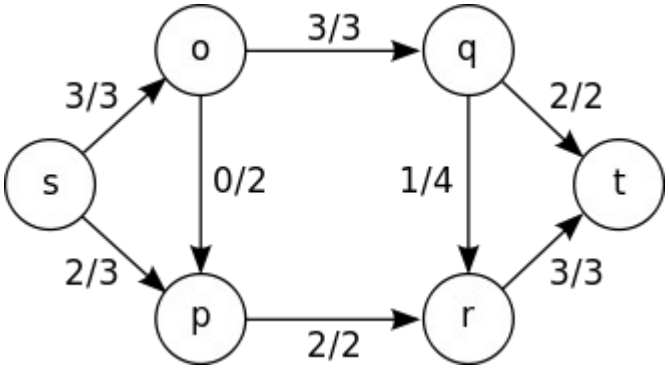
Студент гр. 8302 _____ Халитов Ю.Р.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург
2020

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 проталкивания предпотока.

Описание алгоритма решения.

Для начала проинициализируем предпоток. Пропустим максимально возможный поток по рёбрам, инцидентным истоку, увеличив избыточный поток для каждой смежной с истоком вершиной на соответствующую величину. Все остальные потока не несут, следовательно, для вершин не смежных с истоком избыточный поток изначально будет нулевым. Также для всех вершин, кроме, естественно, истока, установим высоту, равную нулю.

После инициализации будем выполнять операции проталкивания и подъёма в произвольном порядке. Утверждается, что количество данных

операций конечно, и после завершения работы алгоритма наш предпоток является максимальным потоком.

В качестве структуры для хранения графа был написан пользовательский тип Graph, который хранит граф в виде вектора с вершинами и вектора с ребрами. Решение в пользу векторов, а не массивов было принято в силу более удобной работы с памятью у векторов. Идея хранить граф как матрицу смежности была отброшена в связи с тем, что у вершин и ребер имеется множество дополнительных полей, необходимых для реализации алгоритма.

Оценки временной сложности.

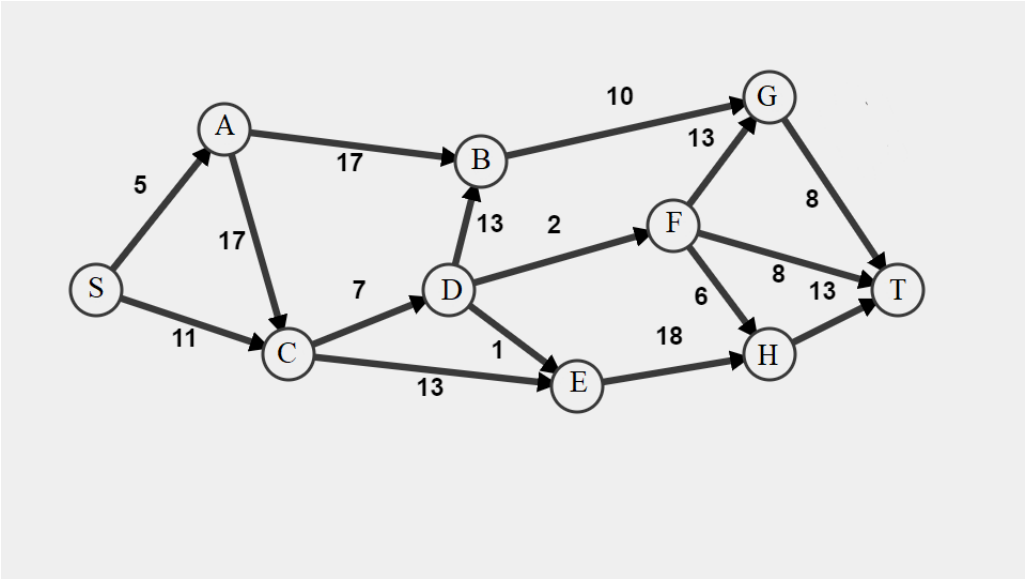
Таблица 1 – Оценки временной сложности методов класса Graph

Метод	Сложность
void preflow()	$O(E)$
int overflow_vertex()	$O(V)$
void update_reverse_edge(int i, int flow)	$O(E)$
bool push(int u)	$O(E)$
void relabel(int u)	$O(E)$
int max_flow()	$O(V^2 E)$

V – количество узлов в графе, E – количество ребер в графе

Примеры работы.

1.




```
Enter csv file name: data0.txt
Maximum flow: 16

C:\Users\yulian\source\repos\cw\Debug\cw.exe (процесс 11712) завершает работу с кодом 0.
```

data0.txt:

S A 5

S C 11

A C 17

A B 17

C D 7

D B 13

C E 13

D E 1

D F 2

B G 10

F G 13

E H 18

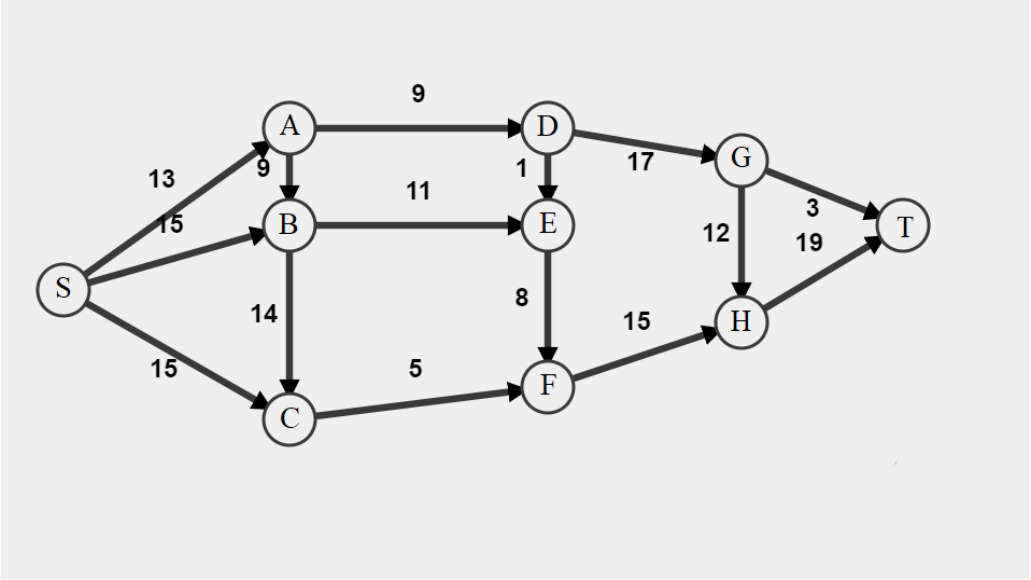
F H 6

G T 8

F T 8

H T 13

2.




```
Enter csv file name: data1.txt
Maximum flow: 22

C:\Users\yulian\source\repos\cw\Debug\cw.exe (процесс 14248) завершает работу с кодом 0.
```

data1.txt:

S A 13

S B 15

S C 15

A B 9

B C 14

A D 9

B E 11

C F 5

D E 1

E F 8

D G 17

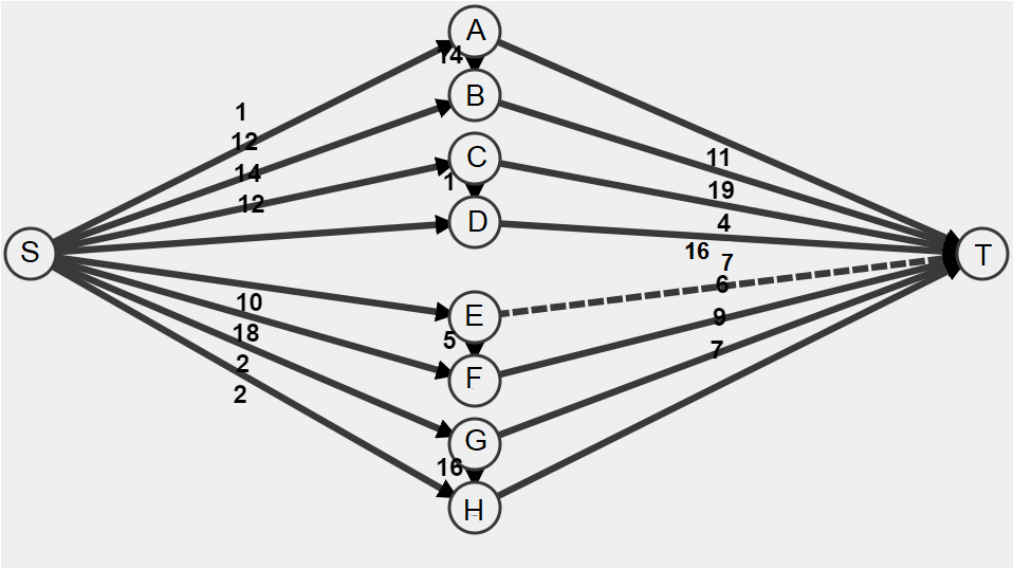
F H 15

G H 12

G T 3

H T 19

3.




```
Enter csv file name: data2.txt
Maximum flow: 47

C:\Users\yulian\source\repos\cw\Debug\cw.exe (процесс 19480) завершает работу с кодом 0.
```

data2.txt:

S A 1

S B 12

S C 14

S D 12

S E 10

S F 18

S G 2

S H 2

A B 14

C D 1

E F 5

G H 16

A T 11

B T 19

C T 4

D T 16

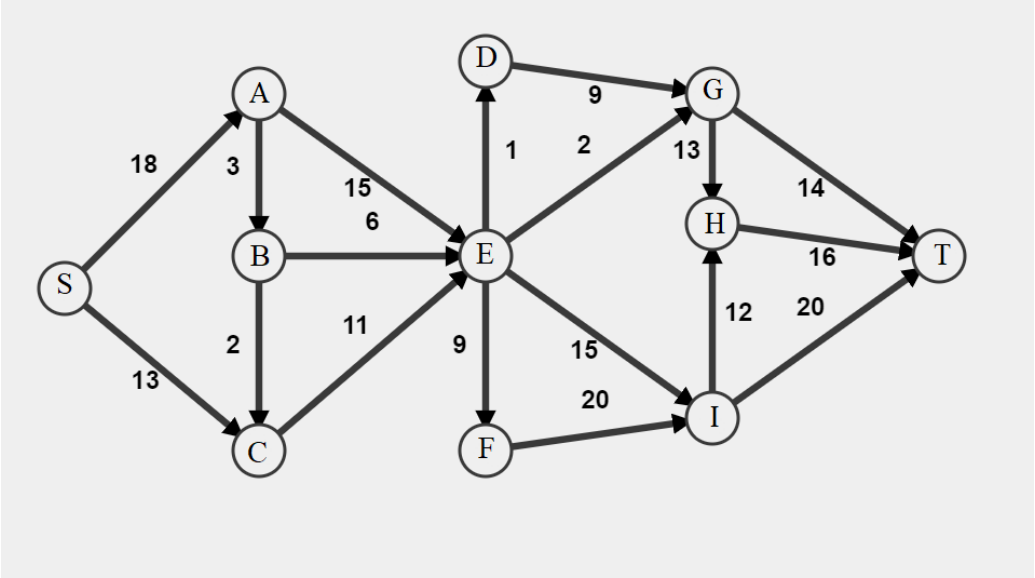
E T 7

F T 6

G T 9

H T 7

4.




```
Enter csv file name: data3.txt
Maximum flow: 27

C:\Users\yulian\source\repos\cw\Debug\cw.exe (процесс 19764) завершает работу с кодом 0.
```

data3.txt:

S A 18

S C 13

A B 3

B C 2

A E 15

B E 6

C E 11

E D 1

E F 9

D G 9

F I 20

E I 15

E G 2

G H 13

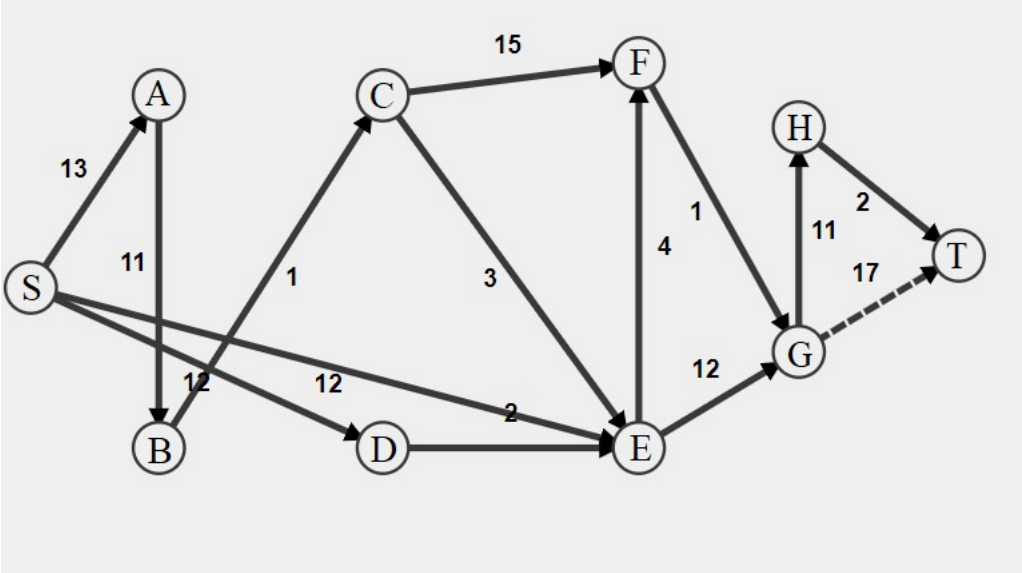
I H 12

G T 14

H T 16

I T 20

5.




```
Enter csv file name: data4.txt
Maximum flow: 13

C:\Users\yulian\source\repos\cw\Debug\cw.exe (процесс 18964) завершает работу с кодом 0.
```

data4.txt:

S A 13

S E 12

S D 12

A B 11

B C 1

D E 2

C E 3

C F 15

E F 4

F G 1

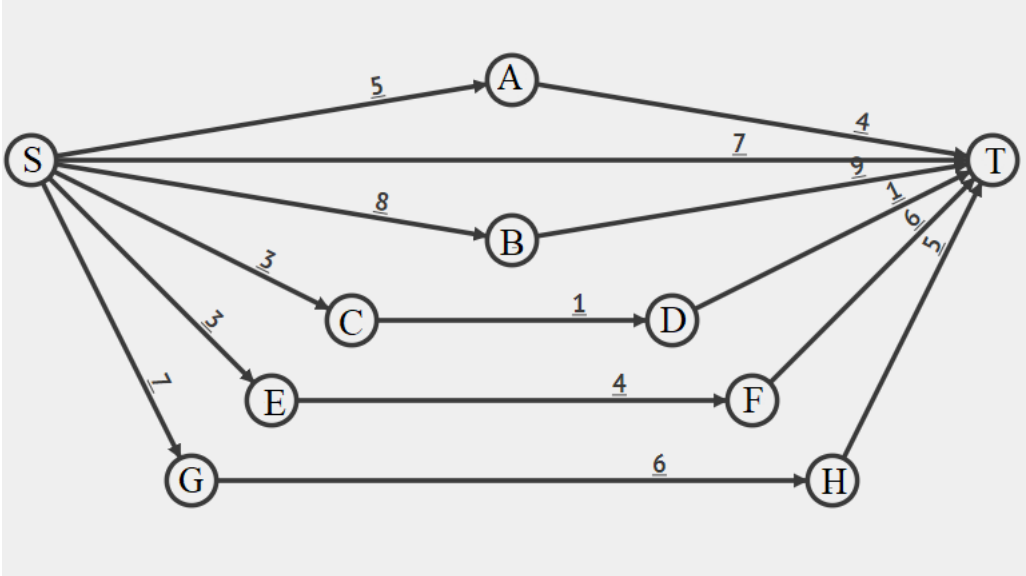
E G 12

H G 11

H T 2

G T 17

6.




```
Enter csv file name: data5.txt
Maximum flow: 28

C:\Users\yulian\source\repos\cw\Debug\cw.exe (процесс 368) завершает работу с кодом 0.
```

data5.txt:

S A 5

A T 4

S T 7

S B 8

B T 9

S C 3

C D 1

D T 1

S E 3

E F 4

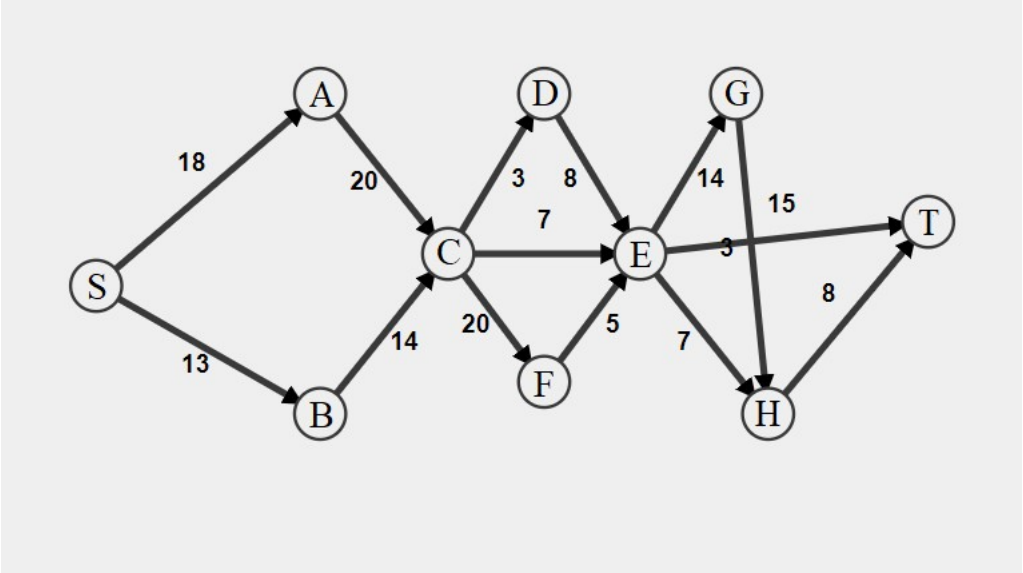
F T 6

S G 7

G H 6

H T 5

7.




```
Enter csv file name: data6.txt
Maximum flow: 15

C:\Users\yulian\source\repos\cw\x64\Release\cw.exe (процесс 9584) завершает работу с кодом 0.
```

data6.txt:

S A 18

S B 13

A C 20

B C 14

C D 3

C F 20

C E 7

D E 8

F E 5

E G 14

G H 3

E H 7

E T 15

H T 8

Листинг.

Graph.h:

```
#pragma once

#include <vector>
#include <algorithm>
#include <string>
#include <map>
#include <sstream>

struct Vertex
{
	int h, excess_flow;
	Vertex(int h, int excess_flow) : h{ h }, excess_flow{
excess_flow } {}
};

struct Edge
{
	int flow, capacity, u, v;
	Edge(int flow, int capacity, int u, int v) : flow{ flow },
capacity{ capacity }, u{ u }, v{ v } {}
};

class Graph
{
	std::vector<Vertex> vertexes;
	std::vector<Edge> edges;
	bool push(int u);
	void relabel(int u);
	void preflow();
	int overflow_vertex();
	void update_reverse_edge(int i, int flow);
public:
	Graph(std::vector<std::string>);
	int max_flow();
};
```

Graph.cpp:

```
#include "Graph.h"

Graph::Graph(std::vector<std::string> rows)
{
	std::map<char, int> vertex_map;
	for (auto row : rows) vertex_map[row[0]] = vertex_map[row[2]]
= 0;
	vertex_map['T'] = vertex_map.size() - 1;
	int i = 1;
	for (auto& vert : vertex_map)
	{
		if (vert.first != 'S' && vert.first != 'T') vert.second
= i++;
		vertexes.push_back(Vertex(0, 0));
	}
	for (auto row : rows)
```

```
     {
          std::stringstream capacity_stream(row.substr(4,
row.length() - 1));
          int capacity = 0;
          capacity_stream >> capacity;
          edges.push_back(Edge(0, capacity, vertex_map[row[0]],
vertex_map[row[2]]));
     }
}


void Graph::preflow()
{
     vertexes[0].h = vertexes.size();
     for (int i = 0; i < edges.size(); ++i)
     {
          if (edges[i].u == 0)
          {
               edges[i].flow = edges[i].capacity;
               vertexes[edges[i].v].excess_flow += edges[i].flow;
               edges.push_back(Edge(-edges[i].flow, 0, edges[i].v,
0));
          }
     }
}


int Graph::overflow_vertex()
{
     for (int i = 1; i < vertexes.size() - 1; ++i)
          if (vertexes[i].excess_flow > 0) return i;
     return -1;
}


void Graph::update_reverse_edge(int i, int flow)
{
     int u = edges[i].v;
     int v = edges[i].u;

     for (int j = 0; j < edges.size(); ++j)
     {
          if (edges[j].v == v && edges[j].u == u)
          {
               edges[j].flow -= flow;
               return;
          }
     }
     edges.push_back(Edge(0, flow, u, v));
}


bool Graph::push(int u)
{
     for (int i = 0; i < edges.size(); ++i)
     {
          if (edges[i].u == u)
          {
               if (edges[i].flow == edges[i].capacity) continue;
```

```
                if (vertexes[u].h > vertexes[edges[i].v].h)
                {
                     int flow = std::min(edges[i].capacity -
edges[i].flow, vertexes[u].excess_flow);
                     edges[i].flow += flow;
                     vertexes[u].excess_flow -= flow;
                     vertexes[edges[i].v].excess_flow += flow;
                     update_reverse_edge(i, flow);
                     return true;
                }
           }
     }
     return false;
}


void Graph::relabel(int u)
{
     int min_h = INT_MAX;
     for (int i = 0; i < edges.size(); ++i)
     {
           if (edges[i].u == u)
           {
                if (edges[i].flow == edges[i].capacity)
                     continue;
                if (vertexes[edges[i].v].h < min_h)
                {
                     min_h = vertexes[edges[i].v].h;
                     vertexes[u].h = min_h + 1;
                }
           }
     }
}

int Graph::max_flow()
{
     preflow();
     while (overflow_vertex() != -1)
     {
           int u = overflow_vertex();
           if (!push(u))
                relabel(u);
     }
     return vertexes.back().excess_flow;
}
```

Functions.h:

```
#pragma once

#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>

bool is_number(std::string);
std::vector<std::string> read_data(std::fstream&);
```

Functions.cpp:

```
#include "Functions.h"

bool is_number(std::string str)
{
     for (char c : str)
          if (!isdigit(c)) return false;
     return true;
}

std::vector<std::string> read_data(std::fstream& file)
{
     std::vector<std::string> rows;
     std::string row;
     while (getline(file, row))
     {
          if (row.length() >= 5 && isalpha(row[0]) && row[1] == '
' &&
               isalpha(row[2]) && row[3] == ' ' &&
is_number(row.substr(4, row.length() - 1)))
          {
               rows.push_back(row);
          }
          else
          {
               rows.clear();
               break;
          }
     }
     return rows;
}
```

cw.cpp:

```
#include "Graph.h"
#include "Functions.h"

using namespace std;


int main()
{
     string file_name;
     cout << "Enter csv file name: ";
     cin >> file_name;
     fstream file;
     file.open(file_name, ios::in);
     if (!file.fail())
     {
          vector<string> rows = read_data(file);
          if (!rows.empty())
          {
               Graph g(rows);
               cout << "Maximum flow: " << g.max_flow();
          }
          else cout << "Wrong input!\n";
          file.close();
```

```
    }
    else cout << "File doesn't exist!\n";
}
```

cw_tests.cpp:

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../cw/Graph.h"
#include "../cw/Functions.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace cwtests
{
    TEST_CLASS(cwtests)
    {
    public:

        TEST_METHOD(max_flow_1)
        {
            std::vector<std::string> rows{ "S A 5", "S C 11",
"A C 17", "A B 17", "C D 7",
                "D B 13", "C E 13", "D E 1", "D F 2", "B G
10", "F G 13",
                "E H 18", "F H 6", "G T 8", "F T 8", "H T
13" };
            Graph g(rows);
            Assert::IsTrue(g.max_flow() == 16);
        }
        TEST_METHOD(max_flow_2)
        {
            std::vector<std::string> rows{ "S A 13", "S E 12",
"S D 12", "A B 11", "B C 1", "D E 2",
                "C E 3", "C F 15", "E F 4", "F G 1", "E G 12",
"H G 11", "H T 2", "G T 17" };
            Graph g(rows);
            Assert::IsTrue(g.max_flow() == 13);
        }
        TEST_METHOD(read_data_success)
```

```
		{
			std::fstream file;
			file.open("test_data_0.txt", std::ios::in);
			std::vector<std::string> rows = read_data(file);
			file.close();
			std::cout << "hey";
			std::vector<std::string> true_rows{ "S T 5", "S A
11", "A S 17" };
			Assert::IsTrue(rows == true_rows);

		}
		TEST_METHOD(read_data_fail)
		{
			std::fstream file;
			file.open("test_data_1.txt", std::ios::in);
			std::vector<std::string> rows = read_data(file);
			file.close();
			Assert::IsTrue(rows.empty());
		}
		TEST_METHOD(is_number_success)
		{
			Assert::IsTrue(is_number("12847187"));
		}
		TEST_METHOD(is_number_fail)
		{
			Assert::IsFalse(is_number("random text"));
		}
	};
}
```