

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: Реализация алгоритма Форда-Фалкерсона
Вариант задания: 1

Студент гр. 8301

Пчёлко В. А.

Преподаватель

Тутуева А.В.

Санкт-Петербург

2020

Содержание пояснительной записки:

1. Задание на курсовую работу
2. Аннотация
3. Оглавление
4. Цель работы
5. Ход работы
6. Постановка задачи
7. Реализация алгоритма
8. Пример работы программы
9. Описание unit-тестов
10. Код программы
11. Описание алгоритма

Предполагаемый объем пояснительной записки:

Не менее 20 страниц

Дата выдачи задания _____

Дата сдачи работы: _____

Дата защиты работы: _____

Студент _____

Пчёлко В.А.

Преподаватель _____

Тутуева А. В.

**ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ**

Студент Пчёлко В.А.

Группа 8302

Тема работы: Реализация алгоритма Форда-Фалкерсона

Исходные данные:

Файл с набором строк, задающий сеть.

АННОТАЦИЯ

Данная курсовая работа проверяет умение работать с сетями. Требуется реализовать алгоритм Форда-Фалкерсона для нахождения максимального потока в сети.

SUMMARY

This coursework tests the ability to work with networks and flows. It is required to implement Ford-Fulkerson algorithm to find max flow in network.

Оглавление

Оглавление	5
Цель работы	6
1 Ход работы	6
1.1 Постановка задачи.....	6
1.2 Выбор структур данных	6
1.3 Обоснование выбора структур данных.....	7
1.4 Реализация алгоритма	8
1.5 Примеры работы программы	9
1.6 Описание реализованных unit-тестов.....	13
2 Код программы	16
Выводы	19

Цель работы

Реализовать алгоритм Форда-Фалкерсона для поиска максимального потока в сети.

1 Ход работы

1.1 Постановка задачи

Необходимо проверить файл на корректность входных данных. Далее считать данные из файла и применить к ним реализованный алгоритм Форда-Фалкерсона.

Входные данные:

- Файл формата:

A...Z A...Z NUM
...
A...Z A...Z NUM

NUM – целое, неотрицательное число

Выходные данные:

- Сообщение Max flow is MAX_FLOW
MAX_FLOW – результат работы алгоритма Форда-Фалкерсона

1.2 Выбор структур данных

Я создал пользовательский класс NetworkFlow. Класс представляет сеть в виде графа и содержит метод – алгоритм Форда-Фалкерсона. Класс содержит в себе поля:

- numOfVertex – число ребер
- numOfEdge – число верши
- sourceVertex – источник сети
- destinationVertex – сток сети
- capacity – одномерный динамический массив пропускных способностей
- onEnd - одномерный динамический массив
- nextEdge - одномерный динамический массив вершин идущих после вершины по номеру которой обратились
- firstEdge - одномерный динамический массив начал рёбер
- visited – одномерный динамический массив для проверки посещённых вершин
- INF – бесконечность – реализована значением INT_MAX величина необходимая для корректно работы алгоритма

Методы класса:

- Конструктор – Вызывает метод CheckInput, инициализирует поля класса, использует метод addEdge. Сложность инициализации $O(V)$, V – количество вершин. Ввиду того что количество полей класса равно const временная сложность инициализации определяется как $O(V)$. Сложность методов CheckInput и addEdge описана ниже.

- Деструктор – удаляет динамические поля класса, работает за $O(const)$ ввиду использования стандартной функции языка delete.
- max_flow – поиск максимального потока в сети. Работает за $O(MAX_FLOW * E)$, обусловлена сходимостью алгоритма только для целочисленных потоков. В случае с нецелочисленными данными сложность алгоритма является бесконечной, не сходясь к правильному решению. В нашем случае проверка ввода не пропустит нецелочисленные данные на вход алгоритма. Метод использует метод findFlow.
- addEdge – метод, используемый конструктором – присваивает полям класса значения, работает за $O(const)$ ввиду обращения к конкретным индексам конкретных полей.
- findFlow – Находит увеличивающий путь, пропускает через него максимальный поток, а также работает с остаточной сетью в соответствии с алгоритмом. Работает за $O(E)$, E – количество ребер графа.
- CheckInput – проверяет корректность входных данных на соответствие шаблону. Кидает исключение в случае ошибки. Также подсчитывает количество вершин в графе. Временная сложность проверки $O(n)$, n – количество строк в файле. Временная сложность подсчета количества вершин V – количество вершин в графе $O(V * \log(V))$ – обусловлена использованием ассоциативного массива, реализованного на основе красно-черного дерева для хранения считанных данных ($O(\log(V))$ – сложность вставки и поиска в красно-черном дереве). Таким образом, в худшем случае, алгоритму придется применить операции поиска по 2 раза и вставки по 1 разу, что даст сложность $(O(V * (2\log(V) + \log(V)))) = O(V * \log(V))$. Альтернативная временная сложность – $O(n * \log(n))$, n – количество строк во входном файле. Следует отметить, что худший случай для этого алгоритма – граф, вырожденный в “ломанную” линию.

1.3 Обоснование выбора структур данных

Использование ассоциативного массива для проверки ввода обусловлено вычислительной сложностью алгоритма. Ввиду того, что ассоциативный массив реализован на основе красно-черного дерева операции вставки и поиска имеют логарифмическую вычислительную сложность, что ускоряет быстроедействие программы на больших объемах данных. Для сравнения: операция поиска в массиве/списке имеет сложность $O(n)$, а в красно-черном дереве $O(\log n)$.

Разработанный мной пользовательский класс для реализации алгоритма обеспечивает объектно-ориентированный подход к задаче. Использование динамических массивов внутри класса обеспечивает оптимальное использование памяти при работе алгоритма. Также представление графов на

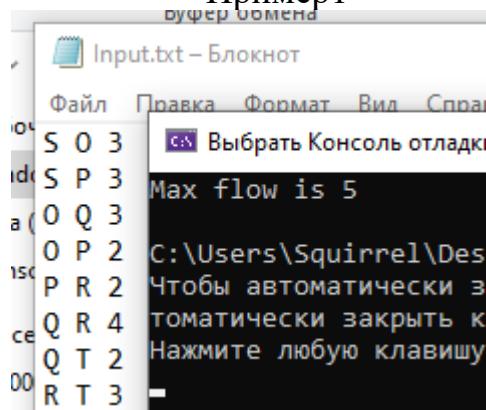
массивах позволяет точно и однозначно описать граф, а следовательно и реализовать алгоритмы к нему.

1.4 Реализация алгоритма

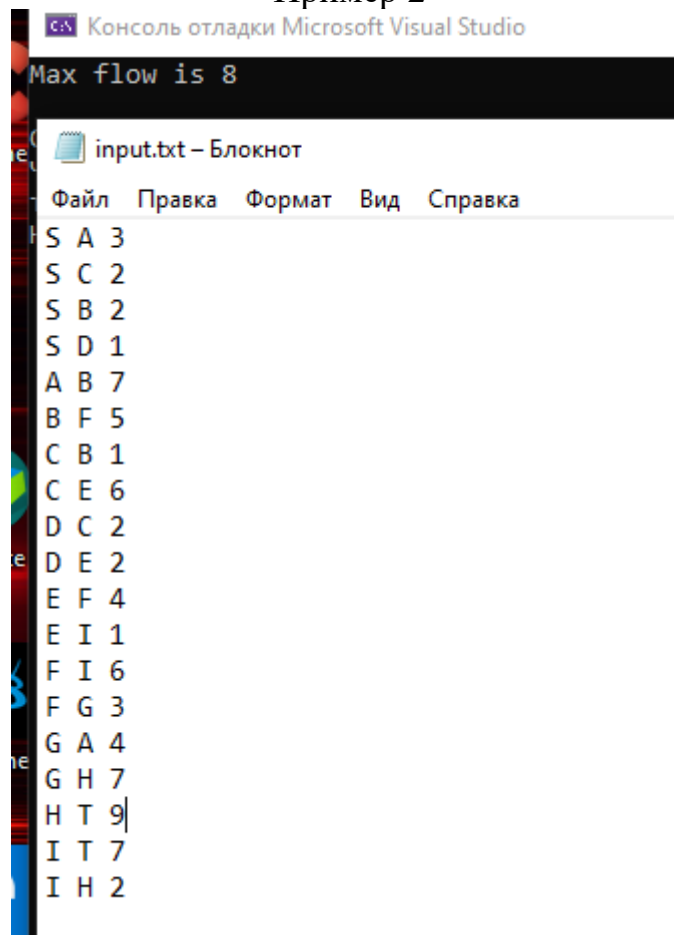
На вход программе поступает файл, после чего он проверяется на корректность входных данных. Далее запускается алгоритм Форда-Фалкерсона.

1.5 Примеры работы программы

Пример 1



Пример 2



Пример 3

```
Консоль отладки Microsoft Visual Studio
Max flow is 87

input.txt - Блокнот
Файл  Правка  Формат  Вид  Справка
S A 43
S C 52
S B 20
S D 12
A B 70
B F 59
C B 10
C E 16
D C 90
D E 12
E F 40
E I 10
F I 60
F G 33
G A 40
G H 70
H T 90
I T 72
I H 20
```

Пример 4

```
Консоль отладки Microsoft Visual Studio
Max flow is 25

input.txt - Блокнот
Файл  Правка  Формат  Вид  Справка
S B 12
S C 32
A B 5
A D 6
B D 3
B E 6
C D 9
C G 23
D E 7
D F 4
E I 17
F H 34
F I 8
G D 2
G F 9
H G 34
H I 22
H T 89
I T 12
```

Пример 5

Консоль отладки Microsoft Visual Studio

A data-entry error. Check the correctness of the input in the file and correct these errors in the line under the number: 1
 C:\Users\Squirrel\Desktop\Ford_Fulkerson\Debug\Ford_Fulkerson.exe (процесс 20276) завершил работу с кодом 0.
 Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрывать консоль при остановке отладки".
 Нажмите любую клавишу, чтобы закрыть это окно...

input.txt – Блокнот

Файл Правка Формат Вид Справка

```
S B
S C 32
A B 5
A D 6
B D 3
B E 6
C D 9
C G 23
D E 7
D F 4
E I 17
F H 34
F I 8
G D 2
G F 9
H G 34
H I 22
H T 89
I T 12
```

Пример 6

Консоль отладки Microsoft Visual Studio

Max flow is 6

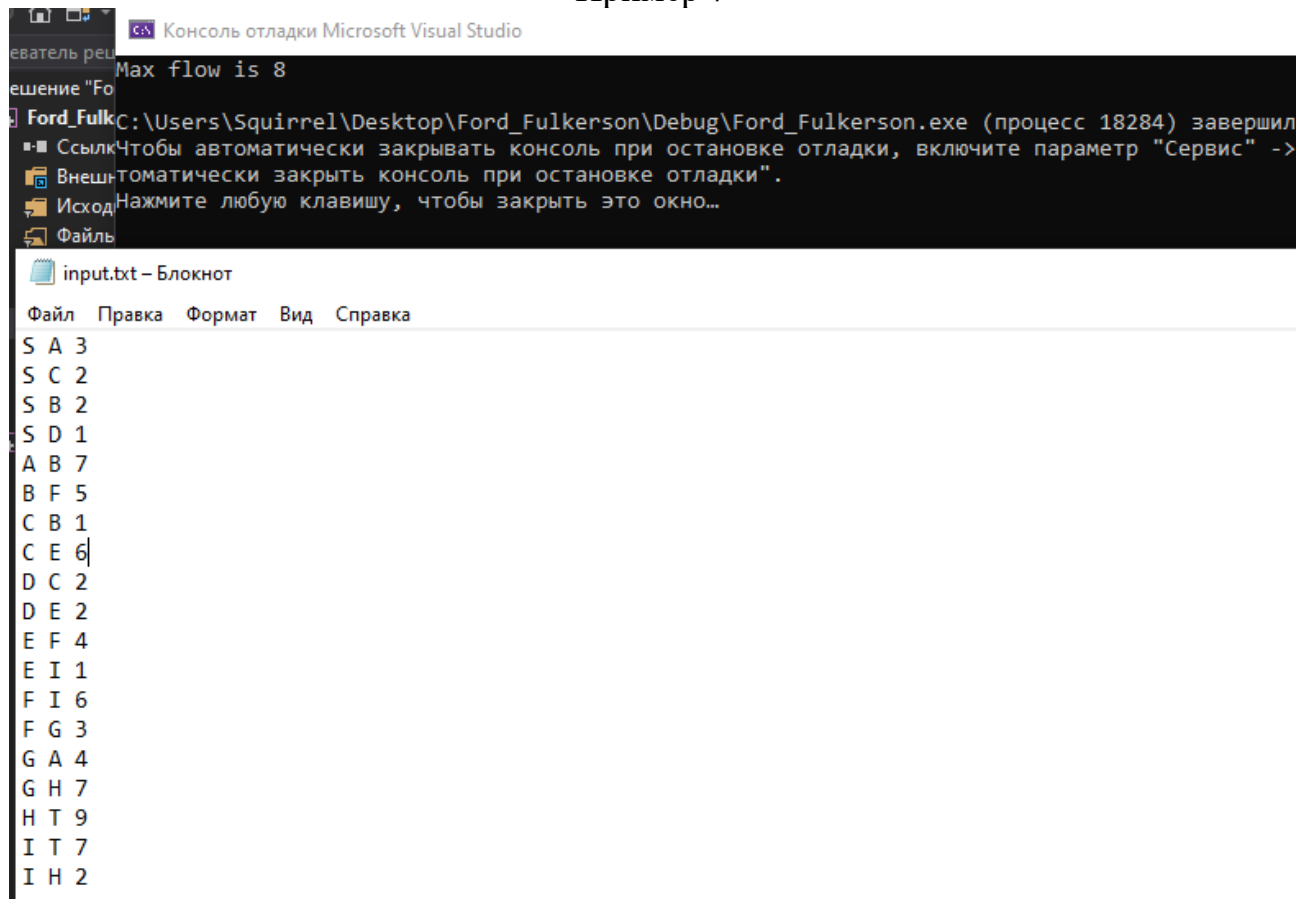
C:\Users\Squirrel\Desktop\Ford_Fulkerson\Debug\Ford_Fulkerson.exe (процесс 20276) завершил работу с кодом 0.
 Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрывать консоль при остановке отладки".
 Нажмите любую клавишу, чтобы закрыть это окно...

input.txt – Блокнот

Файл Правка Формат Вид Справка

```
S A 8
S E 5
A D 6
A B 5
C I 4
D E 5
S C 9
D F 10
E G 5
E I 7
C M 9
A E 3
F L 6
B D 12
F K 2
H M 11
I M 10
K H 7
L T 13
```

Пример 7



The screenshot shows the Microsoft Visual Studio interface. The top window is the 'Консоль отладки Microsoft Visual Studio' (Visual Studio Debug Console). It displays the output of a program: 'Max flow is 8'. Below this, a message indicates that the process 'Ford_Fulkerson.exe' (PID 18284) has finished. A context menu is open over the console, with options: 'Ссылка' (Link), 'Внешний' (External), 'Исходный код' (Source Code), and 'Файл' (File). The 'Внешний' option is selected, showing a message: 'Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Автоматически закрыть консоль при остановке отладки". Нажмите любую клавишу, чтобы закрыть это окно...' (To automatically close the console when debugging stops, enable the parameter "Service" -> "Automatically close console when debugging stops". Press any key to close this window...). The bottom window is a text editor titled 'input.txt - Блокнот' (input.txt - Notepad). It contains a list of edges and their capacities: 'A 3', 'C 2', 'B 2', 'D 1', 'A B 7', 'B F 5', 'C B 1', 'C E 6', 'D C 2', 'D E 2', 'E F 4', 'E I 1', 'F I 6', 'F G 3', 'G A 4', 'G H 7', 'H T 9', 'I T 7', 'I H 2'.

```
Max flow is 8

C:\Users\Squirrel\Desktop\Ford_Fulkerson\Debug\Ford_Fulkerson.exe (процесс 18284) завершил
Ссылка
Внешний
Исходный код
Файл

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->
"Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

input.txt - Блокнот

Файл Правка Формат Вид Справка

S A 3
S C 2
S B 2
S D 1
A B 7
B F 5
C B 1
C E 6
D C 2
D E 2
E F 4
E I 1
F I 6
F G 3
G A 4
G H 7
H T 9
I T 7
I H 2

1.6 Описание реализованных unit-тестов

Юнит тесты проверяют как работу кода на корректных данных так и обработку исключений.

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../Ford_Fulkerson/NetworkFlow.h"
#include <fstream>
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace FordFulkersonAlgoTests
{
    TEST_CLASS(FordFulkersonAlgoTests)
    {
    public:

        TEST_METHOD(TestMethod_Correct_output_for_6_vertexes)
        {
            ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input1.txt");
            NetworkFlow Flows(input);
            Assert::AreEqual(Flows.max_flow(), 5);
        }
        TEST_METHOD(TestMethod_Exception_entering_the_first_character) {
            try {
                ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input2.txt");
                NetworkFlow Flows(input);
            }
            catch (exception& ex) {
                Assert::AreEqual(ex.what(), "Error entering the first character
in the string or missing a space after it. Check the correctness of the input in the file
and correct these errors in the line under the number: 2");
            }
        }
        TEST_METHOD(TestMethod_Exception_entering_the_second_character) {
            try {
                ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input3.txt");
                NetworkFlow Flows(input);
            }
            catch (exception& ex) {
                Assert::AreEqual(ex.what(), "Error entering the second character
in the string or missing a space after it. Check the correctness of the input in the file
and correct these errors in the line under the number: 2");
            }
        }
        TEST_METHOD(TestMethod_Exception_entering_the_third_number_Flows) {
            try {
                ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input4.txt");
                NetworkFlow Flows(input);
            }
            catch (exception& ex) {
                Assert::AreEqual(ex.what(), "Error entering the third character
(bandwidth) in the string or the presence of a space after it. Please note that the bandwidth
cannot be negative. Check that you entered the file correctly and correct these errors in
the line number: 2");
            }
        }
        TEST_METHOD(TestMethod_Exception_empty_string) {
            try {
```

```

        ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input5.txt");
        NetworkFlow Flows(input);
    }
    catch (exception& ex) {
        Assert::AreEqual(ex.what(), "A data-entry error. Check the
correctness of the input in the file and correct these errors in the line under the number:
2");
    }
}

TEST_METHOD(TestMethod_Correct_output_for_6_vertexes_and_edge_from_source_to_sink)
{
    ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input6.txt");
    NetworkFlow Flows(input);
    Assert::AreEqual(Flows.max_flow(), 25);
}

TEST_METHOD(TestMethod_Correct_output_for_2_vertexes_edges_from_source_to_sink)
{
    ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input7.txt");
    NetworkFlow Flows(input);
    Assert::AreEqual(Flows.max_flow(), 20);
}
TEST_METHOD(TestMethod_Exception_there_is_a_path_from_the_vertex_to_itself) {
    try
    {
        ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input8.txt");
        NetworkFlow Flows(input);
    }
    catch (exception& ex) {
        Assert::AreEqual(ex.what(), "The path from the vertex to itself
is impossible in the string under the number: 2");
    }
}
TEST_METHOD(TestMethod_Correct_output_for_20_vertexes)
{
    ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input9.txt");
    NetworkFlow Flows(input);
    Assert::AreEqual(Flows.max_flow(), 19);
}
TEST_METHOD(TestMethod_Exception_missing_source) {
    try
    {
        ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input10.txt");
        NetworkFlow Flows(input);
    }
    catch (exception& ex) {
        Assert::AreEqual(ex.what(), "Source is missing");
    }
}
TEST_METHOD(TestMethod_Exception_missing_sink) {
    try
    {
        ifstream
input("C:\\Users\\Squirrel\\Desktop\\FordFulkersonAlgoTests\\Input11.txt");
        NetworkFlow Flows(input);
    }
    catch (exception& ex) {

```

```

        Assert::AreEqual(ex.what(), "Sink is missing");
    }
}TEST_METHOD(TestMethod_MinFunctionMin){
    int a = 1;
    int b = 2;
    int m = min(a, b);
    Assert::AreEqual(m, 1);
}TEST_METHOD(TestMethod_Exception_MinFunctionEqual) {
    int a = 1;
    int b = 1;
    int m = min(a, b);
    Assert::AreEqual(m, 1);
}
};
}

```

2 Код программы

Ford_Fulkerson.cpp

```
#include <iostream>
#include<fstream>
#include "Map.h"
#include "NetworkFlow.h"
using namespace std;

int main() {

    ifstream input("C:\\Users\\Squirrel\\source\\repos\\Ford_Fulkerson\\input.txt");
    NetworkFlow* flow=new NetworkFlow(input);

    // Выводим максимальный поток
    int max = flow->max_flow();

    input.close();
    delete flow;
    cout << "Max flow is " << max << endl;
    return 0;
}

//string input_string;
//StringHandle(input_string);
return 0;
}
```

NetworkFlow.h

```
#pragma once
#include<fstream>
#include<string>
#include "Map.h"
#include "Min.h"
class NetworkFlow {
public:
    ~NetworkFlow() {
        if (this != NULL) {
            delete[] capacity;
            delete[] onEnd;
            delete[] nextEdge;
            delete[] firstEdge;
            delete[] visited;
        }
    }
    NetworkFlow(ifstream& file) {
        Map<char, int>* Map_from_char_to_number = new Map<char, int>();

        int edgeCount = 0;
        numOfEdge = 0;
        numOfVertex = 0;
        CheckInput(file, Map_from_char_to_number);
        visited = new int[numOfVertex];
        firstEdge = new int[numOfVertex*2];
        capacity = new int[numOfEdge*2];
        onEnd = new int[numOfEdge*2];
        nextEdge = new int[numOfEdge*2];
        for (int i = 0; i < numOfVertex; ++i) {
```



```

        firstEdge[i] = -1;
        visited[i] = 0;
    }

    file.clear();
    file.seekg(ios::beg);

    while (!file.eof()) {
        string s1;
        int vert1, vert2, cap;
        getline(file, s1);
        vert1 = Map_from_char_to_number->find(s1[0]);
        vert2 = Map_from_char_to_number->find(s1[2]);
        cap = stoi(s1.substr(4));
        addEdge(vert1, vert2, edgeCount, cap);
    }
    Map_from_char_to_number->clear();
}

int max_flow() {
    int maxFlow = 0;
    int iterationResult = 0;
    while ((iterationResult = findFlow(sourceVertex, INF)) > 0) {
        for (int i = 0; i < numOfVertex; ++i)
            visited[i] = 0;
        maxFlow += iterationResult;
    }
    return maxFlow;
}

void addEdge(int edge, int vertex, int& edgeCount, int cap) {
    // Прямое ребро
    onEnd[edgeCount] = vertex; // на конце прямого vertex
    nextEdge[edgeCount] = firstEdge[edge]; // добавляем в начало списка для
edge
firstEdge[edge] = edgeCount; // теперь начало списка - новое
ребро
capacity[edgeCount++] = cap; // устанавливаем пропускную
способность
    // Обратное ребро
    onEnd[edgeCount] = edge; // на конце обратного edge
    nextEdge[edgeCount] = firstEdge[vertex]; // добавляем в начало списка для
vertex
firstEdge[vertex] = edgeCount; // теперь начало списка - новое
ребро
capacity[edgeCount++] = 0; // устанавливаем пропускную
способность
}

int findFlow(int edge, int flow) {
    if (edge == destinationVertex)
        return flow; // возвращаем полученный минимум на пути
    visited[edge] = true;
    for (int edges = firstEdge[edge]; edges != -1; edges = nextEdge[edges]) {
        int to = onEnd[edges];
        if (!visited[to] && capacity[edges] > 0) {
            int minResult = findFlow(to, min(flow, capacity[edges])); // ищем
поток в поддереве
            if (minResult > 0) { // если нашли
                capacity[edges] -= minResult; // у прямых ребер вычитаем
поток
                capacity[edges ^ 1] += minResult; // к обратным
прибавляем
            }
        }
    }
    return 0;
}

```

```

        return minResult;
    }
}
}
return 0; // если не нашли поток из этой вершины вернем 0
}

void CheckInput(ifstream& file, Map<char, int>*& Map_from_char_to_number) {
    int str_num = 1;
    while (!file.eof()) {
        string s1;
        getline(file, s1);
        if (s1.size() >= 5) { // больше или равно 5, потому что это минимальный
возможный ввод (две буквы, два пробела, одна цифра)
            if (!((s1[0] >= 'A' || s1[0] <= 'Z') && (s1[1] == ' '))) {
                throw std::exception(string("Error entering the first
character in the string or missing a space after it. Check the correctness of the input in
the file and correct these errors in the line under the number: " +
to_string(str_num))).c_str());
            }
            if (!((s1[2] >= 'A' || s1[2] <= 'Z') && (s1[3] == ' '))) {
                throw std::exception(string("Error entering the second
character in the string or missing a space after it. Check the correctness of the input in
the file and correct these errors in the line under the numbe: " +
to_string(str_num))).c_str());
            }
            string cur;

            for (int i = 4; i < s1.size(); ++i) {
                if (s1[i] >= '0' || s1[i] <= '9')
                    cur += s1[i];
                else {
                    throw std::exception(string("Error entering the
third character (bandwidth) in the string or the presence of a space after it. Check the
correctness of the input in the file and correct these errors in the line under the number:"
+ to_string(str_num))).c_str());
                }
            }
            if (!Map_from_char_to_number->is_in_map(s1[0])) { // если нету
тогда добавляем инкрементируем кол-во вершин
                Map_from_char_to_number->insert(s1[0], numOfVertex);
                ++numOfVertex;
            }
            if (!Map_from_char_to_number->is_in_map(s1[2])) { // если нету
тогда добавляем инкрементируем кол-во вершин
                Map_from_char_to_number->insert(s1[2], numOfVertex);
                ++numOfVertex;
            }

        }
        else
        {
            throw std::exception(string("A data-entry error. Check the
correctness of the input in the file and correct these errors in the line under the number:
" + to_string(str_num))).c_str());
        }
        ++str_num;
        ++numOfEdge;
    }

    if (Map_from_char_to_number->is_in_map('S'))
        sourceVertex = Map_from_char_to_number->find('S');
    else {
        throw std::exception("Source is missing");
    }
}

```

```

        }

        if (Map_from_char_to_number->is_in_map('T'))
            destinationVertex = Map_from_char_to_number->find('T');
        else {
            throw std::exception("Sink is missing");
        }
    }
private:
    int numOfVertex;
    int numOfEdge;
    int sourceVertex;
    int destinationVertex;

    int* capacity;
    int* onEnd;
    int* nextEdge;
    int* firstEdge;
    int* visited;

    int INF = INT_MAX;
};

```

Min.h

```

#pragma once
template<typename T>
T min(T a, T b) {
    return a > b ? b : a;
}

```

Выводы

Реализовал алгоритм Форда-Фалкерсона. Закрепил навыки в ООП на языке C++.