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
Алгоритм Форда-Фалкерсона

Вариант 1

Студент 0301 _____ Сморжок В. Е.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург,

2022

Задание: 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

Текст программы:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int invalidPrice = INT_MAX;

string read()
{
	string text = "";
	string newText;
	ifstream creat;
	char ch;
	creat.open("Net.txt");

	while (!creat.eof()) {
		creat.get(ch);

		text += ch;
	}
	creat.close();
	newText = text.substr(0, text.length() - 1);
	return newText;
}

class List
{
private:
	struct Node
	{
		string name;
		Node* next = NULL;
	};
	Node* head;
	Node* tail;
```

```
public:
	List()
	{
		head = tail = NULL;
	}

	void add(string name)
	{
		if (head == NULL)
		{
			Node* buffer = new Node;
			buffer->name = name;
			buffer->next = NULL;
			head = tail = buffer;
		}
		else
		{
			Node* buffer = head;
			Node* prev = head;
			while (buffer != NULL)
			{
				prev = buffer;
				buffer = buffer->next;
			}
			buffer = new Node;
			buffer->name = name;
			buffer->next = NULL;
			prev->next = buffer;
			tail = buffer;
		}
	}
	bool check(string text)
	{
		Node* buffer = head;
		bool i = false;
		while (buffer != NULL)
		{
			if (buffer->name == text)
			{
				i = true;
				break;
			}
			buffer = buffer->next;
		}
		return i;
	}
	void print()
	{
		Node* buffer = head;
		while (buffer != NULL)
		{
			cout << buffer->name << endl;
			buffer = buffer->next;
		}
	}
	Node* stringToList(string text)
	{
		int i = 0;
		string stringText = "";
		while (i != text.length())
		{
			while (text[i] != ' ')
			{
				if (text[i] <= 90 && text[i] >= 65 || text[i] <= 57 && text[i]
>= 48)
```

```
                        {
                                stringText += text[i];
                        }
                        if (text[i] == '\n')
                        {
                                stringText = "";
                        }
                        i++;
                        if (i == text.length())
                        {
                                break;
                        }
                }
                if (i == text.length())
                {
                        break;
                }
                i++;
                if (!check(stringText))
                        add(stringText);
                stringText = "";
        }
        Node* buffer = head;
        return buffer;
}
int** creatingMatrix(string text)
{
        int i = 0;
        int** array = new int* [0];
        array = new int* [maxIndex() + 1];
        for (int count = 0; count <= maxIndex(); count++)
                array[count] = new int[maxIndex()];
        for (int count_row = 0; count_row <= maxIndex(); count_row++)
                for (int count_column = 0; count_column <= maxIndex();
count_column++)
                        array[count_row][count_column] = invalidPrice;
        i = 0;
        string stringText = "";
        string vertex1 = "";
        int index1 = 0;
        string vertex2 = "";
        int index2 = 0;
        int data1 = 0;
        int data2 = 0;
        while (1)
        {
                while (text[i] != ' ')
                {
                        if (text[i] <= 90 && text[i] >= 65 || text[i] <= 57 && text[i]
>= 48)
                        {
                                stringText += text[i];
                        }

                        if (text[i] == '\n')
                        {
                                break;
                        }
                        i++;
                        if (i == text.length())
                        {
                                break;
                        }
                }
                if (vertex1 == "")
```

```
                        {
                                vertex1 = stringText;
                                index1 = index(vertex1);
                                stringText = "";
                        }
                        else if (vertex2 == "")
                        {
                                vertex2 = stringText;
                                index2 = index(stringText);
                                stringText = "";
                        }
                        if (stringText != vertex1 && stringText != vertex2 && stringText !=
""")
                        {
                                data1 = stoi(stringText);
                                data2 = 0;
                                stringText = "";
                        }
                        if (data1 != 0 && index1 != -1 && index2 != -1)
                        {
                                array[index1][index2] = data1;
                                array[index2][index1] = data2;
                                data1 = data2 = 0;
                                index1 = index2 = -1;
                                vertex1 = vertex2 = "";
                        }
                        i++;
                        if (i >= text.length())
                                break;
                }
                cout << "Initial network matrix:" << endl;
                printMatrix(array);
                return array;
        }
        string find(int index)
        {
                Node* buffer = head;
                int i = 0;
                while (buffer != NULL)
                {
                        if (i == index)
                        {
                                break;
                        }
                        else {
                                buffer = buffer->next;
                                i++;
                        }
                }
                if (buffer != NULL)
                        return buffer->name;
        }
        void printMatrix(int** arrayNew)
        {
                for (int i = 0; i <= maxIndex(); i++)
                {
                        for (int j = 0; j <= maxIndex(); j++)
                        {
                                if (arrayNew[i][j] != invalidPrice)
                                        cout << arrayNew[i][j] << " ";
                                else cout << "N ";
                        }
                        cout << endl;
                }
                cout << endl;
```

```
	}
	int index(string text)
	{
		Node* buffer = head;
		int i = 0;
		while (buffer != NULL)
		{
			if (buffer->name == text)
			{
				break;
			}
			buffer = buffer->next;
			i++;
		}
		return i;
	}
	int maxIndex()
	{
		Node* buffer = head;
		int i = 0;
		while (buffer != NULL)
		{
			buffer = buffer->next;
			i++;
		}
		return i - 1;
	}
};

int AlghoritmFordFalkerson(string graph)
{
	List list;
	list.stringToList(graph);
	int stream = invalidPrice;
	int maxF = 0;
	int check = false;
	int check2 = false;
	int** checkMatrix = new int* [0];
	checkMatrix = new int* [list.maxIndex() + 1];
	for (int count = 0; count <= list.maxIndex(); count++)
		checkMatrix[count] = new int[list.maxIndex()];
	for (int count_row = 0; count_row <= list.maxIndex(); count_row++)
		for (int count_column = 0; count_column <= list.maxIndex(); count_column++)
			checkMatrix[count_row][count_column] = 0;
	int num = 0;
	bool check3 = false;
	int** array = list.creatingMatrix(graph);

	for (int i = 0; i < list.maxIndex(); i++)
	{
		for (int j = 0; j < list.maxIndex(); j++)
		{
			stream = invalidPrice;
			int numI = i;
			int numJ = j;
			while (j != list.maxIndex() + 1)
			{
				if (array[i][j] != invalidPrice && array[i][j] != 0 &&
checkMatrix[i][j] == 0)
				{
					if (array[i][j] < stream)
						stream = array[i][j];
					check = true;
					num = i;
					i = j;
```

```
                                        j = 0;
                                }
                                else j++;
                        }
                        if (check == false)
                        {
                                check2 = true;
                                break;
                        }
                        i = numI;
                        j = numJ;
                        while (j != list.maxIndex() + 1)
                        {
                                if (array[i][j] != invalidPrice && array[i][j] != 0 &&
checkMatrix[i][j] == 0)
                                {
                                        num = array[i][j];
                                        array[i][j] -= stream;
                                        array[j][i] += stream;
                                        checkMatrix[j][i] = 1;
                                        num = i;
                                        i = j;
                                        j = 0;
                                }
                                else j++;
                        }
                        maxF += stream;
                        i = 0;
                        j = list.maxIndex();
                        int count = 0;
                        while (i != list.maxIndex() + 1)
                        {
                                if (array[i][j] == 0)
                                        count++;
                                else
                                        if (array[i][j] != invalidPrice)
                                        {
                                                num = array[i][j];
                                        }
                                i++;
                        }
                        if (count != 0)
                        {
                                maxF += num;
                                check2 = true;
                                break;
                        }
                        i = -1;
                        check = false;
                }
                if (check2 == true)
                {
                        check3 = true;
                        break;
                }
        }

        cout << "Residual network matrix:" << endl;
        list.printMatrix(array);
        cout << "Maximum network flow = " << maxF << endl;
        return maxF;
}


int main()
```

```
{
	string graph = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
	cout << graph << endl;
	AlghoritmFordFalkerson(graph);
	return 0;
}
```

Текст Unit-тестов:

```
#include "pch.h"
#include "..\CourseWork\Header.h"
#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestCourseWork
{
	TEST_CLASS(UnitTestCourseWork)
	{
	public:

		TEST_METHOD(TestMethodDoMatrix)
		{
			string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T
3";
			List list;
			list.stringToList(text);
			int** array = list.creatingMatrix(text);
			int arrayTest[6][6] = {
				{invalidPrice, 3, 3, invalidPrice,invalidPrice,invalidPrice},
				{0, invalidPrice, 2, 3, invalidPrice,invalidPrice },
				{0, 0, invalidPrice, invalidPrice, 2, invalidPrice},
				{invalidPrice, 0 ,invalidPrice, invalidPrice, 4, 2},
				{invalidPrice, invalidPrice, 0 ,0 ,invalidPrice ,3},
				{invalidPrice ,invalidPrice ,invalidPrice, 0, 0, invalidPrice}
			};
			for (int i = 0; i < 6; i++)
			{
				for (int j = 0; j < 6; j++)
				{
					Assert::IsTrue(arrayTest[i][j] == array[i][j]);
				}
			}

		}
		TEST_METHOD(TestMethodMaxIndex)
		{
			string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T
3";
			List list;
			list.stringToList(text);
			Assert::IsTrue(list.maxIndex() == 5);
		}
		TEST_METHOD(TestMethodIndex)
		{
			string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T
3";
			List list;
			list.stringToList(text);
			Assert::IsTrue(list.index("S") == 0);
			Assert::IsTrue(list.index("O") == 1);
			Assert::IsTrue(list.index("P") == 2);
			Assert::IsTrue(list.index("Q") == 3);
			Assert::IsTrue(list.index("R") == 4);
			Assert::IsTrue(list.index("T") == 5);
```

```
		}
		TEST_METHOD(TestMethodFind)
		{
			string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T
3";
			List list;
			list.stringToList(text);
			Assert::IsTrue(list.find(0) == "S");
			Assert::IsTrue(list.find(1) == "O");
			Assert::IsTrue(list.find(2) == "P");
			Assert::IsTrue(list.find(3) == "Q");
			Assert::IsTrue(list.find(4) == "R");
			Assert::IsTrue(list.find(5) == "T");
		}
		TEST_METHOD(TestMethodFordFalkerson)
		{
			string graph = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T
3";
			Assert::IsTrue(AlghoritmFordFalkerson(graph)==5);
		}
	};
}
```

Описание реализуемых алгоритмов:

1. stringToList – функция, которая обрабатывает поданную на вход строку, выписывает все названия вершин в односвязный список без повторений с помощью метода add(). Для исключения повторений названий вершин графа была создана функция check().
2. maxIndex() – обрабатывается строка, вызывается функция maxIndex, которая возвращает максимальный индекс вершины. Он сравнивается с заданным индексом. Это и является размерностью матрицы+1
3. print() – печатает список вершин без повторений
4. find – находит элемент списка по индексу и возвращает его имя
5. printMatrix– печатает матрицу, поданную на вход
6. AlghoritmFordFalkerson– получает на вход строку. Создает односвязный список, который заполняется, конвертируется в матрицу. Матрица заполняется, анализируя значения строки. Матрица обрабатывается и возвращается значение максимального потока сети
7. creatingMatrix – создается матрица с помощью обработки односвязного списка
8. index – возвращает индекс элемента по названию
9. printMatrix – печатает поданную на вход матрицу
10. check – проверяет по названию, есть ли в односвязном списке элемент

Описание Unit-тестов:

1. TestMethodFordFalkerson – вводится строка, которая обрабатывается алгоритмом Форда-Фалкерсона, который преобразует ее в матрицу и обрабатывает ее, возвращая значение максимального потока сети. Это значение сравнивается с константным

2. TestMethodDoMatrix – вводится строка, которая обрабатывается, а затем вводится в матрицу, которая впоследствии сравнивается с заданной матрицей

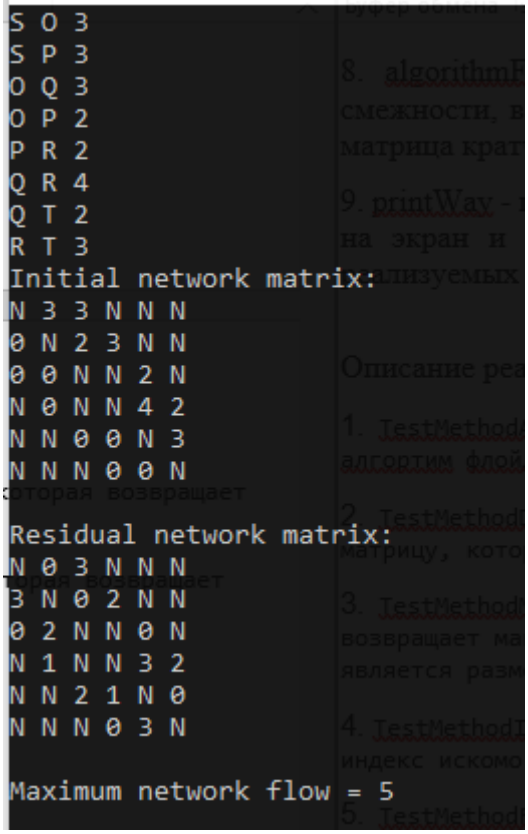
3. TestMethodMaxIndex - обрабатывается строка, вызывается функция maxIndex, которая возвращает максимальный индекс вершин. Он сравнивается с заданным индексом. Это и является размерностью матрицы+1

4. TestMethodIndex - обрабатывается строка, вызывается функция index, которая возвращает индекс искомой вершины. Он сравнивается с заданным индексом

5. TestMethodFind – обрабатывается строка, вызывается функция find, которая возвращает название искомой вершины. Он сравнивается с заданным названием

Оценка временной сложности каждого метода:

1. stringToList – O(n^2)
2. maxIndex() – O(n)
3. print() – O(n)
4. find – O(n)
5. printMatrix– O(n^2)
6. AlghoritmFordFalkerson– O(n^3)
7. creatingMatrix – O(n^2)
8. index – O(n)
9. printMatrix – O(n^2)
10. check – O(n)



```
TEST_CLASS(UnitTestCourseWork)
{
public:

    TEST_METHOD(TestMethodDoMatrix)
    {
        string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
        List list;
        list.stringToList(text);
        int** array = list.creatingMatrix(text);
        int arrayTest[6][6] = {
            {invalidPrice, 3, 3, invalidPrice,invalidPrice,invalidPrice},
            {0, invalidPrice, 2, 3, invalidPrice,invalidPrice },
            {0, 0, invalidPrice, invalidPrice, 2, invalidPrice},
            {invalidPrice, 0 ,invalidPrice, invalidPrice, 4, 2},
            {invalidPrice, invalidPrice, 0 ,0 ,invalidPrice ,3},
            {invalidPrice ,invalidPrice ,invalidPrice, 0, 0, invalidPrice}
        };
        for (int i = 0; i < 6; i++)
        {
            for (int j = 0; j < 6; j++)
            {
                Assert::IsTrue(arrayTest[i][j] == array[i][j]);
            }
        }

    }
    TEST_METHOD(TestMethodMaxIndex)
    {
        string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
        List list;
        list.stringToList(text);
```

```
TEST_METHOD(TestMethodMaxIndex)
{
    string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
    List list;
    list.stringToList(text);
    Assert::IsTrue(list.maxIndex() == 5);
}
TEST_METHOD(TestMethodIndex)
{
    string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
    List list;
    list.stringToList(text);
    Assert::IsTrue(list.index("S") == 0);
    Assert::IsTrue(list.index("O") == 1);
    Assert::IsTrue(list.index("P") == 2);
    Assert::IsTrue(list.index("Q") == 3);
    Assert::IsTrue(list.index("R") == 4);
    Assert::IsTrue(list.index("T") == 5);
}
TEST_METHOD(TestMethodFind)
{
    string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
    List list;
    list.stringToList(text);
    Assert::IsTrue(list.find(0) == "S");
    Assert::IsTrue(list.find(1) == "O");
    Assert::IsTrue(list.find(2) == "P");
    Assert::IsTrue(list.find(3) == "Q");
    Assert::IsTrue(list.find(4) == "R");
    Assert::IsTrue(list.find(5) == "T");
}
```

```
            Assert::IsTrue(list.index("S") == 0);
            Assert::IsTrue(list.index("O") == 1);
            Assert::IsTrue(list.index("P") == 2);
            Assert::IsTrue(list.index("Q") == 3);
            Assert::IsTrue(list.index("R") == 4);
            Assert::IsTrue(list.index("T") == 5);
        }
        TEST_METHOD(TestMethodFind)
        {
            string text = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
            List list;
            list.stringToList(text);
            Assert::IsTrue(list.find(0) == "S");
            Assert::IsTrue(list.find(1) == "O");
            Assert::IsTrue(list.find(2) == "P");
            Assert::IsTrue(list.find(3) == "Q");
            Assert::IsTrue(list.find(4) == "R");
            Assert::IsTrue(list.find(5) == "T");
        }
        TEST_METHOD(TestMethodFordFalkerson)
        {
            string graph = "S O 3\nS P 3\nO Q 3\nO P 2\nP R 2\nQ R 4\nQ T 2\nR T 3";
            Assert::IsTrue(AlghoritmFordFalkerson(graph)==5);
        }
    };
}
```