**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: Алгоритм Форда-Фалкесона

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Пискунов Я.А. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

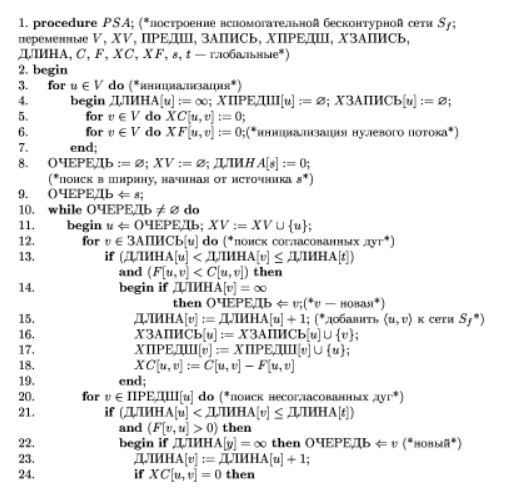
**Цель работы.**

Изучить и реализовать алгоритм Форда-Фалкерсона на языке C++

**Основные теоретические положения.**

*Алгоритм Форда-Фалкерсона* – алгоритм нахождения максимального потока в сети (взвешенном графе).

За основу возьмем алгоритм на языке Paskal, представленный в книге Липского «Комбинаторика для программистов»:



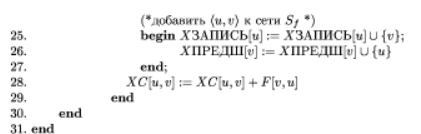


Рисунок 1- псевдокод алгоритма

В данной работе используем структуру представления графа, которую мы создали для предыдущей работы.

**Ход работы**

Граф хранится с помощью двух структур. В первой структуре хранятся элементы графа, а во второй – его сыновья. Каждая структура содержит указатель на своего представителя в графической сцене. Ребра содержат не только указатель на второй элемент, но и его имя. Это связано с тем, что при вводе из файла может быть дана ссылка на вершину, которой ещё нет в основном списке. В таком случае записывается только имя, и зависимости разрешаются позднее. В отличие от предыдущей лабораторной, теперь ребра поддерживают веса. Кроме того, пометки на ребрах теперь могут быть разных порядков. Структуры представлены в приложении А.

Для удобства работы с графом создадим класс, в который поместим все необходимые переменные и структуры, а также методы работы с ним. Заголовочный файл с данными по этому классу представлен в приложении Б. Рассматривать подробно детали реализации графа в данном отчете не будем, так как они не являются главной задачей данной работы. Комментарии к реализации указанных методов есть непосредственно в коде.

Помимо всего прочего, класс включает в себя макросы для сохранения и восстановления итераторов:

#define SAVEITS List\* t\_lpos = KeepItL(); Elem\* t\_pos = KeepItE(); int t\_linpos = linpos;

#define RESTOREITS RestoreItL(t\_lpos); RestoreItE(t\_pos); linpos = t\_linpos;

Они нужны для того, чтобы запуск методов класса не изменял итераторы, запущенные на уровень выше.

Интерфейс программы оставим без изменений. Лишь добавим кнопку запуска алгоритма. Его вид представлен на рис. 2. Аспекты работы с интерфейсом изложены внутри самой программы и вызываются по нажатию кнопки F1 или выбора пункта Help в меню About. Они изложены в виде набора html-страниц, по которым можно перемещаться.

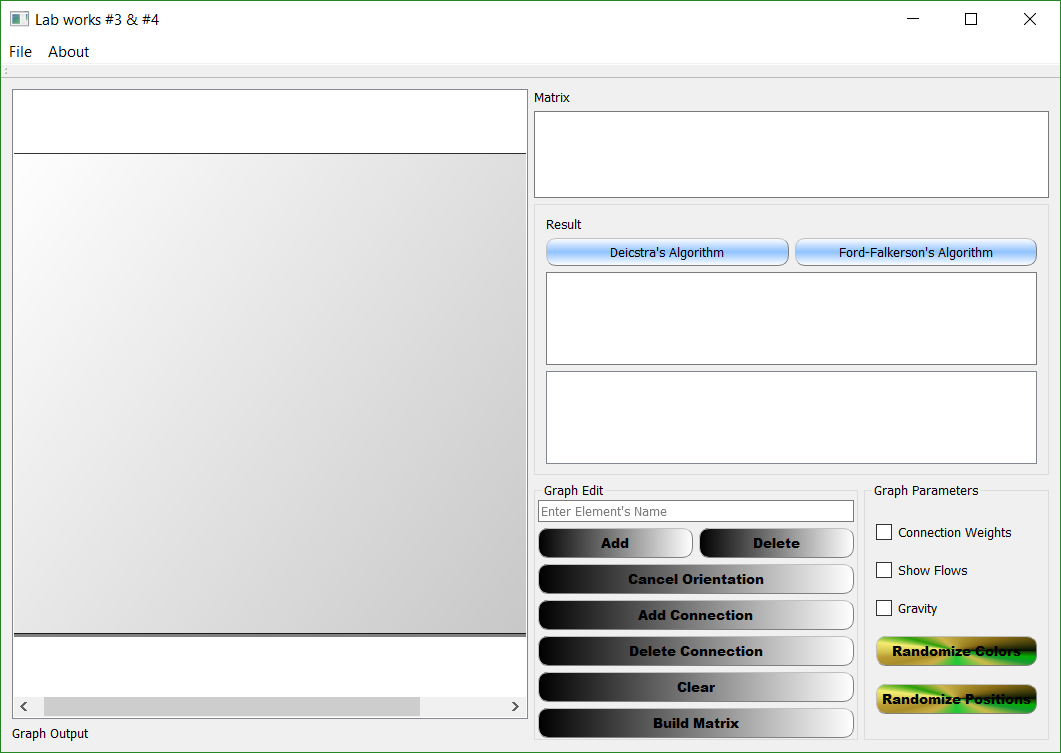


Рисунок 2 – интерфейс программы

Программа может как считывать готовый граф из файла определенной структуры, так и работать с созданным с нуля с помощью QGraphicsScene графом. Нюансы работы с ним также описаны в меню Help. Программа поддерживает возможность редактирования графа, добавление и удаление вершин и ребер из него. Кроме этого, в ней реализованы два алгоритма. В данной работе внимание уделим алгоритму Форда-Фалкерсона. Прежде чем запустить алгоритм, необходимо выбрать исходную вершину(исток) и конечную вершину(сток), относительно которых будет работать алгортим. Запуск алгоритма осуществляется по нажатию соответствующей кнопки в интерфейсе. Код, который при этом выполняется представлен в приложении В.

Так как при выполнении алгоритма необходимо взаимодействовать с графом, то целесообразно добавить алгоритм как один из публичных методов в класс графа. Сам алгоритм состоит из двух частей: инициализация с проверкой возможности выполнения алгоритма и непосредственно сам алгоритм. Эти части соответственно сделаны в виде отдельных методов. Они представлены в приложении Г.

Также необходимо как-то выводить результат на экран. Формат вывода результата следующий. Так как максимальный поток есть сумма нескольких потоков от истока к стоку, то каждый из таких потоков выводится в качестве результата следующим образом: <Величина потока>:<Последовательность вершин, через которые проходит поток>. В конце также выводится суммарный поток, который является максимальным. Кроме этого, каждый путь выделяется отдельным цветом для наглядности. Это можно будет увидеть в продемонстрированных примерах ниже.

Также в интерфейсе предусмотрена функция показа величины потока, проходящего через каждое ребро. Она показывает их после прохождения алгоритма. До этого поток через все ребра равен нулю.

Теперь проведем несколько тестов программы. Для этого возьмем пример из все той же книги Липского «Комбинаторика для программистов», также один простой пример. Результаты работы программы представлены на рис. 3 и рис. 4 (исток обозначен красным, а сток – синим).

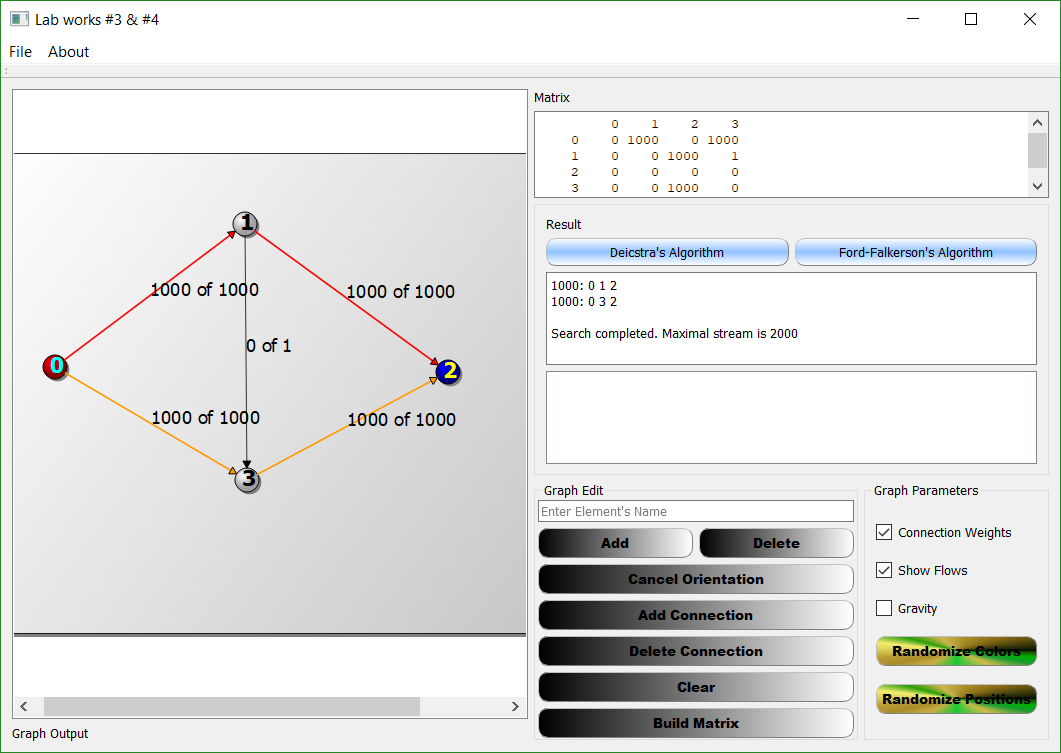


Рисунок 3 – пример работы программы

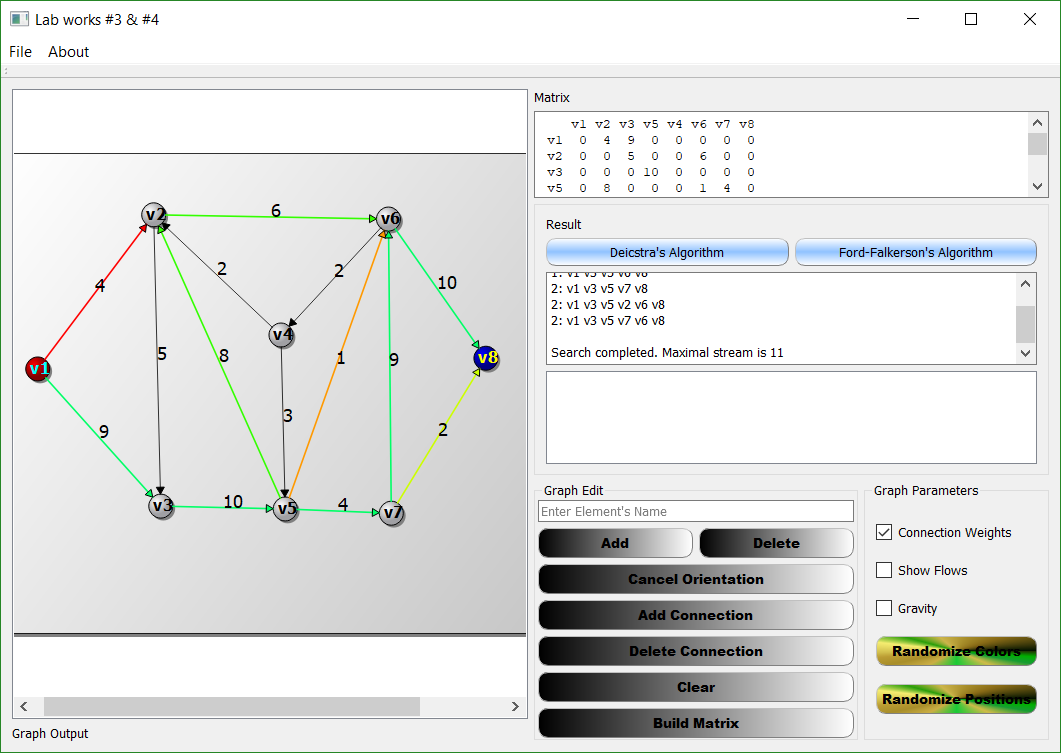


Рисунок 4 – пример работы программы

Как мы видим, результаты работы программы соответствуют представленным в книге и здравому смыслу, что говорит о правильности построения алгоритма.

**Выводы.**

При выполнении данной работы изучен алгоритм Форда-Фалкерсона и его реализация на языке C++.

Приложение А

Структура графа

typedef struct Elem{

Elem() = default;

char name[Numb];

Elem\* next=nullptr;

List\* childs=nullptr;

Node\* node;

bool searched = false;

}Elem;

typedef struct List{

List() = default;

char name[Numb];

int mark = 0;

int weight = 1;

int flow = 0;

Elem\* node = nullptr;

List\* next = nullptr;

Edge\* edge;

}List;

Приложение Б

Код graph.H

#ifndef GRAPH\_H

#define GRAPH\_H

#define SAVEITS List\* t\_lpos = KeepItL(); Elem\* t\_pos = KeepItE(); int t\_linpos = linpos;

#define RESTOREITS RestoreItL(t\_lpos); RestoreItE(t\_pos); linpos = t\_linpos;

#define Numb 80

#define Wid 800

#define Hei 600

#include <iostream>

#include <cstring>

#include <climits>

#include <QTextStream>

#include <QString>

#include <QFile>

#include <QStack>

#include <QMessageBox>

#include <QInputDialog>

#include <QTime>

#include "graphwidget.h"

using namespace std;

typedef struct List List;

typedef struct Elem{

Elem() = default;

char name[Numb];

Elem\* next=nullptr;

List\* childs=nullptr;

Node\* node;

bool searched = false;

}Elem;

typedef struct List{

List() = default;

char name[Numb];

int mark = 0;

int weight = 1;

int flow = 0;

Elem\* node = nullptr;

List\* next = nullptr;

Edge\* edge;

}List;

typedef struct Path{

QList<List\*> way;

QList<Elem\*> way\_el;

int min\_flow = INT\_MAX / 2;

}Path;

class Graph

{

public:

Graph();

~Graph();

void **Clear**();

Elem\* **it**();

List\* **it**(Elem\* el, bool marked = 1);

List\* **itin**(Elem\* el);

void **ResetIts**();

void **ReadFile**(QString fileName);

void **SaveFile**(QString fileName, bool pos = 0);

char\* **GetMinStupidName**();

char\* **GetLastStupidName**();

void **AddElem**(char\* name, double X = -100, double Y = -100);

void **AddEdge**(Elem\* el1, Elem\* el2, int weight = 1);

void **AddEdge**(Elem\* el1, char\* name, int weight = 1);

bool **Solve**();

void **RemoveElem**(char\* name);

void **RemoveEdge**(Elem\* el1, Elem\* el2);

void **RemoveEdges**(Elem\* el);

void **RenameElem**(char\* oldname, char\* newname);

void **Desorientate**();

Elem\* **FindElem**(char\* name);

int **CountChildren**(Elem\* el, bool marked = 1);

int **CountElems**();

int **Is\_Egde**(Elem\* el1, Elem\* el2, bool noabs = 0);

List\* **GetEdge**(Elem\* el1, Elem\* el2);

int **Max\_Width**();

void **Inc\_Matr**(QTextStream& os);

void **Mark**(List\* ls, int mark = 0);

void **ClearMarks**();

Elem\* **FindPath**(Elem\* el);

void **ResetFindPath**();

void **SetMinPath**();

bool **DeicstraInit**(Elem \*v0i = nullptr);

void **DeicstraAlgorithm**();

QQueue<List\*> **MakeQueue**(Elem\*\* previous);

Elem\* **FindMinimal**(QQueue<List \*> \*H);

void **FordInit**();

bool **FordAlgorithm**();

void **WeightsOn**(bool state);

void **ChangeWeight**(Elem\* el1, Elem\* el2, int weight);

double **AverageWeight**();

void **ClearWeights**();

Elem\* **operator**[](int i);

int **number**(Elem\* el);

GraphWidget\* widget;

Elem\* marked = nullptr;

bool itermarks = false;

bool weights = false;

Elem\* source = nullptr;

Elem\* sink = nullptr;

Path path;

int max\_iter = 0;

Elem\* v0 = nullptr;

int\* arr = nullptr;

int ib = 0;

int vm = 10;

int changes = 0;

bool go = 1;

private:

int it\_num = 0;

int stupidnames = 0;

Elem\* gr;

Elem\* pos;

List\* lpos;

int linpos;

void **Clear**(List\* ls, Elem \*el);

void **Clear**(Elem\* gr);

Elem\* **KeepItE**();

List\* **KeepItL**();

void **RestoreItE**(Elem\* t\_pos);

void **RestoreItL**(List\* t\_lpos);

};

#endif // GRAPH\_H

Приложение В

Код запуска алгоритма

void MainWindow::**on\_Ford\_clicked**()

{

bool result;

int sum = 0;

while ((result = gr1->FordAlgorithm()))

{

if (result)

{

QString temp;

QTextStream strm(&temp);

strm << gr1->path.min\_flow << ": ";

sum += gr1->path.min\_flow;

for (auto it : gr1->path.way\_el)

strm << it->name << " ";

strm << "\n";

QString temp2 = ui->ResultEd->toPlainText();

temp2.append(temp);

ui->ResultEd->setPlainText(temp2);

}

}

ui->ResultEd->append("Search completed. Maximal stream is " + QString::number(sum));

}Приложение Г

Код Методов алгоритма

Elem\* Graph::**FindPath**(Elem \*el)

{

SAVEITS;

List\* ls;

if (el->searched)

{

RESTOREITS;

return nullptr;

}

Elem\* temp;

el->searched = 1;

path.way\_el.push\_back(el);

if (el == sink){

RESTOREITS;

return el;

}

while ((ls = it(el))!=nullptr)

{

if ((!ls->node->searched) && (ls->weight - ls->flow > 0))

{

path.way.push\_back(ls);

ls->mark = it\_num;

ls->edge->update();

temp = FindPath(ls->node);

if (temp == sink){

RESTOREITS;

return temp;

}

path.way.pop\_back();

ls->mark = 0;

ls->edge->update();

}

}

path.way\_el.pop\_back();

RESTOREITS;

return nullptr;

}

void Graph::**ResetFindPath**()

{

SAVEITS;

Elem\* el;

path.way.clear();

path.way\_el.clear();

path.min\_flow = INT\_MAX/2;

while ((el = it())!=nullptr){

el->searched = 0;

}

RESTOREITS;

}

void Graph::**SetMinPath**()

{

int res = INT\_MAX / 2;

for (auto it : path.way){

if ((it->weight - it->flow) < res)

res = it->weight - it->flow;

}

path.min\_flow = res;

}

void Graph::**FordInit**()

{

if (!sink || !source)

{

QMessageBox box;

box.setText("No source or sink marked");

box.*exec*();

return;

}

if (!weights)

{

QMessageBox box;

box.setText("No weights");

box.*exec*();

return;

}

max\_iter = 10;

}

bool Graph::**FordAlgorithm**()

{

if (max\_iter == 0)

FordInit();

if (max\_iter == 0)

return false;

it\_num++;

ResetFindPath();

FindPath(source);

SetMinPath();

if (path.way.size() == 0)

return false;

int i = 0;

for (auto current : path.way)

{

current->flow = current->flow + path.min\_flow;

Elem\* temp = path.way\_el.at(i++);

List\* templist = GetEdge(current->node, temp);

if (templist)

templist->flow = current->flow;

}