МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему **Сравнительный анализ алгоритмов решения задачи о максимальном потоке**

Студент Н.А. Божинов _____

Руководитель М.А. Тренина _____

Консультант по аннотации Н.В. Ященко _____

Допустить к защите
Заведующий кафедрой к.т.н., доцент А.В. Очеповский _____

«_____» _____ 20___ г.

Тольятти 2017

Аннотация

Темой данной бакалаврской работы является «Сравнительный анализ алгоритмов решения задачи о максимальном потоке».

Работа выполнена студентом Тольяттинского государственного университета, института математики, физики и информационных технологий, группы ПМИб-1301, Божинова Никиты Алексеевича.

Объект работы: алгоритмы решения задачи о максимальном потоке.

Предмет исследования: сравнительный анализ реализаций алгоритмов решения задачи о максимальном потоке на языке программирования C++.

Цель работы: реализовать рассмотренные алгоритмы решения задачи о максимальном потоке, выявить их достоинства и недостатки.

Для достижения цели работы необходимо решить следующие задачи:

1) Рассмотреть основные алгоритмы программирования на сетях.

2) Реализовать распространенные и востребованные алгоритмы на языке C++.

3) Выявить достоинства каждой из реализаций алгоритмов.

Отчет состоит из введения, трех глав и заключения.

В первой главе представлены основные алгоритмы программирования на сетях.

Во второй главе описано проектирование системы максимального потока на сетях.

В третьей главе описана реализация алгоритмов и производится сравнительный анализ реализаций.

Бакалаврская работа представлена на 45 страницах, включает 19 иллюстраций, 1 приложение, список используемой литературы содержит 29 источников.

Abstract

The title of the graduation work is "The Comparative Analysis of Algorithms for Solving the Problem of the Maximum Flow ".

The aim of the graduation work is to make a comparative analysis of algorithms for solving the maximum flow problem.

The object of the graduation work is the solution of the maximum flow problem.

The subject of this work is the comparative analysis of implementations of algorithms for solving the maximum flow problem in the programming language C++.

This graduation work consists of an explanatory note on 45 pages, three parts, introduction on 2 pages, including 19 figures, 4 tables, a list of 29 references, including 9 sources in a foreign language and 1 appendix.

The graduation work is devoted to the question of the comparative analysis of the algorithms for solving problems of searching for the maximum flow in a network.

We study in details the issue of implementing a unified computer system with the transition to paperless work technology, automatic generation of reporting forms, automated analysis, support and control of decisions.

The key issue in the graduation work is the comparative analysis of the methods of Ford-Falkerson, Edmonds-Carp, Dinitsa algorithm and a new parallel "get to the top" algorithm for obtaining results, using the programming tools Biilder C ++.

All parts are aimed at studying the formulations of network tasks, which at first glance are not networked, but which either directly or with some modifications can be reduced to network tasks. The advantage of this approach is that, when using network designs, the efficiency of computations can be significantly increased.

Comparison of algorithms in this graduation work is based on the results of simulation experiments for various algorithms and configurations. This simulation allows avoiding long-term field experiments and ensuring reproducibility of the results.

The developed system has great practical value, this system makes processing of information faster, improves the quality of information processing, automates the input and output of information.

Оглавление

ВВЕДЕНИЕ

В последнее время в различных областях знаний широко применяется теория графов. С помощью теории графов хорошо описываются задачи экономической и планово-производственной практики, как, например, календарное и сетевое планирование и управление, автоматизация управления производством, рационализация схем перевозок и грузопотоков, оптимальное размещение производства т.п.

Задача поиска максимального потока в сети, в общем случае, имело решение подобное известной на тот момент транспортной задачи. Симплексный метод специально для этой задачи был модифицирован в 1951 году Данцигом. Алгоритм не был полиномиальным, как и любой симплексный метод. Однако специальный вид задачи позволил предположить, что для ее решения можно применить более простые и быстрые алгоритмы. Первый такой алгоритм, который не похож на симплексный метод, был разработан Фордом и Фалкерсоном в 1956 году [1].

Время работы этого алгоритма не было полиномиальным, однако его идея и его математическое обоснование как раз и легли в основу качественной теории потоков. В предложенном методе использовалась идея «дополняющих путей».

Актуальность выпускной квалификационной работы обусловлена необходимостью выбора и автоматизации алгоритма решения задачи о максимальном потоке.

Объектом исследования является задача о максимальном потоке.

Предметом исследования является автоматизация поиска максимального потока.

Цель работы: провести сравнительный анализ различных алгоритмов решения задачи о максимальном потоке.

Задачи:

1) рассмотреть основные алгоритмы программирования на сетях;
2) разработать систему автоматизации поиска максимального потока на сетях;
3) провести сравнительный анализ реализованных алгоритмов.

Отчет состоит из введения, трех глав и заключения.

Глава 1 Основные алгоритмы программирования на сетях

1.1 Сети. Основные определения и теоремы

Сеть - это связный ориентированный граф G(V, E) без петель с выделенными истоком и стоком – I и S соответственно. Заметим, что каждой дуге соответствует некоторое натуральное число $c(v_i; v_j)$ – пропускная способность дуги.

Поток в сети задает способ пересылки или же передачи некоторых объектов из одной вершины графа в другую по заданному направлению дуги. Число объектов $f(v_i; v_j)$, передаваемых вдоль дуги $(v_i; v_j)$, не может быть больше пропускной способности $c(v_i; v_j)$ этой дуги: $0 \leq f(v_i; v_j) \leq c(v_i; v_j)$. Значит, если существует дуга из v_i в v_j, то нет дуги из v_j в v_i [2]. Таким образом, рассматривать поток вещества будем только в одну сторону.

Формулировка задачи о максимальном потоке: на сети с заданными пропускными способностями дуг сформировать максимальный по величине поток F_{max} между ее истоком и стоком. Этот поток обеспечивается назначением в каждой дуге $(v_i; v_j)$ величины $f(v_i; v_j)$ передаваемого ею потока [3].

Задача о максимальном потоке в сети должна удовлетворять следующим условиям:

– сумма потоков дуг, выходящих из истока сети, не изменяется и должна быть равна сумме потоков дуг, входящих в сток

$$\sum_{(I,v_i)\in E} f(I, v_i) = \sum_{(v_j,S)\in E} f(v_j, S);$$

– для вершины v, которая не является стоком или истоком, т.е. $v \neq I, v \neq S$, количество единиц потока, которые входят в вершину, должно быть равно количеству единиц потока, выходящего из нее (сохранение потока)

$$\sum_{(v_i,v)\in E} f(v_i, v) = \sum_{(v,v_j)\in E} f(v, v_j);$$

– максимальный поток на пути от истока I к стоку S определяется той дугой $(v_i; v_j)$, которая имеет наименьшую или же минимальную пропускную способность из всех дуг, принадлежащих этому пути.

Дуга $(v_i; v_j)$ называется насыщенной, если её пропускная способность $c(v_i; v_j)$ равна потоку $f(v_i; v_j)$, идущему через неё, а любой путь, в который она включена, будет называться насыщенным путём.

Поток называется насыщенным, если любой путь из I в S содержит дугу $(v_i; v_j)$, для которой $f\ v_i; v_j\ = c(v_i; v_j)$. Первая часть решения задачи о максимальном потоке состоит в нахождении насыщенного потока. Но насыщенный поток не всегда является максимальным.

Он (поток) в сети будет максимален, только если его величина F_{max} больше величины любых других потоков этой сети [4].

1.2 Разрез на сети

Разрез – это множество дуг, удаление которых из сети сделало бы орграф несвязным.

Допустим, дана некоторая сеть. Разобьем множество вершин V этой сети на два непересекающихся подмножества A и B $(A \cup B = V; A \cap B \neq \emptyset)$ так, чтобы в подмножестве A был исток I, а в подмножестве B – сток S, т.е. $I \in A, S \in B$. В случаях, подобных данному, говорят, что на сети произведен разрез, отделяющий исток I от стока S.

Пусть $R(A/B)$ - разрез на сети, представляющий собой совокупность дуг, связывающих подмножества вершин A и B. В разрез входят дуги, обозначим их R^+, начальные вершины которых принадлежат подмножеству A, а конечные – подмножеству B, т.е. $R^+ = \ v_i; v_j\ |v_i \in \mathrm{A}, v_j \in \mathrm{B}\ $. А также в разрез входят дуги, обозначим их R^-, начальные вершины которых принадлежат подмножеству B, а конечные – подмножеству A, т.е. $R^- = \ v_i; v_j\ |v_i \in \mathrm{B}, v_j \in \mathrm{A}\ $.

Величина $C(A/B)$ называется пропускной способностью или величиной разреза $R(A/B)$ и определяется следующей формулой:

$$C\ A/B\ = C\ R^{+}\ - C\ R^{-}\ = \sum_{v_i \in A, v_j \in B} c\ v_i; v_j\ - \sum_{v_i \in B, v_j \in A} c\ v_i; v_j\ .$$

Потоком через разрез $R(A/B)$ называется величина $F(A/B)$, которая определяется следующей формулой:

$$F\ A/B\ = F\ R^{+}\ - F\ R^{-}\ = \sum_{v_i \in A, v_j \in B} f\ v_i; v_j\ - \sum_{v_i \in B, v_j \in A} f\ v_i; v_j\ .$$

1.3 Алгоритмы нахождения максимального потока на сети

1.3.1 Алгоритм Форда-Фалкерсона

Если на сети сформирован некоторый поток, то для ответа на вопрос: будет ли он максимальным, как правило, используют теорему Форда-Фалкерсона.

Теорема Форда-Фалкерсона.

Максимальный поток в сети равен минимальной пропускной способности по всем разрезам:

$$F_{max} = C_{min}(A/B)$$

Доказательство.

▲ Дан граф G(V,E) с пропускной способностью c(u,v) и потоком f(u,v) = 0 для ребер из u в v. Нужно найти максимальный поток из источника s в сток t. На каждом шаге алгоритма действуют те же условия, что и для всех потоков:

$$f(u,v) \le c(u,v).$$

Поток из u в v не может превосходить пропускную способность.

$$f\ u,v\ = -f(v,u).$$

$$\sum_{v} f\ u,v\ = 0 \Leftrightarrow f_{in}\ u\ = f_{out}\ u$$

для всех узлов u, кроме истока и стока. При прохождении через узел поток остается каким и был.

Остаточная сеть $G_f(V,E_f)$ — сеть с пропускной способностью $c_f(u,v) = c(u,v) - f(u,v)$ и без потока.

Вход: Граф G с пропускной способностью c, источник s и сток t.

Выход: Максимальный поток f из s в t.

1. $f(u,v) \leftarrow 0$ для всех ребер (u,v)
2. Пока есть путь p из s в t в G_f, такой что $c_f\ u,v\ > 0$ для всех ребер $(u,v) \in p$:
    1. Найти $c_f\ p\ = min\ c_f\ u,v \mid u,v\ \in p$
    2. Для каждого ребра $u,v\ \in p$
        1. $f\ u,v\ \leftarrow f\ u,v\ + c_f\ p$
        2. $f\ v,u\ \leftarrow f\ v,u\ - c_f\ p$

Путь может быть найден, например, поиском в ширину (алгоритм Эдмондса-Карпа) или поиском в глубину в $G_f(V,E_f)$ [5].

Сложность

Добавляя поток увеличивающего пути к уже имеющемуся потоку, максимальный поток будет получен, когда нельзя будет найти увеличивающий путь. Тем не менее, если величина пропускной способности — иррациональное число, то алгоритм может работать бесконечно. В целых числах таких проблем не возникает и время работы ограничено $O(Ef)$, где E — число рёбер в графе, f — максимальный поток в графе, так как каждый увеличивающий путь может быть найден за $O(E)$ и увеличивает поток как минимум на 1.

Этап 1. Насыщение потока.

Шаг 1. Сформировать произвольный начальный поток.

Шаг 2. Найти оставшиеся возможные пути из истока I в сток S, имеющие только ненасыщенные дуги. Если такой путь найден, то переходим к шагу 3. Если путь не найден, то переходим к шагу 4.

Шаг 3. Увеличить поток по найденному пути таким образом, чтобы одна из дуг стала насыщенной.

Шаг 4. Получившийся поток насыщен.

Этап 2. Пометка вершин сети. (Перераспределение потока.)

Шаг 5. Вершину I пометить «-I».

Шаг 6. Пусть m – любая из уже помеченных вершин; n – некоторая непомеченная вершина, смежная с вершиной m. Вершину n помечаем $+m$, если данные вершины соединены ненасыщенной дугой $m \to n(+m)$, и помечаем -m, если соединены непустой дугой $m \leftarrow n(-m)$.

После пометки вершин возможны два случая: вершина S оказалась либо помеченной (шаг 7), либо непомеченной (шаг 8).

Шаг 7. Оказалось, вершина S имеет метку. Значит, существует последовательность помеченных вершин от I к S. В этой последовательности каждая последующая вершина помечена буквой предыдущей вершины. На дугах последовательности определяем новый поток. Увеличиваем на δ единиц поток на дугах, имеющих направление от I к S и уменьшаем на δ единиц поток на дугах, имеющих обратное направление. Число δ равно наименьшей разнице между пропускной способностью и потоком дуг, входящих в последовательность. Поток можно увеличивать (уменьшать) на прямых (обратных) дугах настолько, пока одна из дуг не станет насыщенной (пустой). Перераспределение увеличивает поток на δ единиц в вершину S. Далее вновь переходим к пометке вершин (шаг 5).

Этап 3. Определение максимального потока.

Шаг 8. Вершина S осталась непомеченной. Пусть A – множество всех помеченных вершин, B – множество всех непомеченных вершин. Тогда дуги, связывающие два подмножества вершин A и B, определяют разрез $R(A/B)$. Таким образом, найден поток F и разрез $R(A/B)$, для которого выполняется условие $F_{max} = C_{min}(A/B)$.▼

Алгоритм Форда-Фалкерсона

1°. Пронумеровать произвольным образом вершины сети T, отличные от входа x_0 и выхода z.

2°. Построить произвольный поток φ на транспортной сети T (например, положить $\varphi(u)=0, \forall u \in \Gamma$).

3°. Просмотреть пути, соединяющие вход сети x_0 с выходом z. Если поток φ полный, то перейти к пункту 4°. Если нет, то рассмотреть путь μ, соединяющий x_0 с z, все дуги которого не насыщены. Построить новый поток φ':

$$\varphi'(u)=\begin{cases}\varphi(u), \text{если } u \in \bar{\mu},\\ \varphi(u)+k, \text{если } u \in \mu,\end{cases}$$

где $k=\min\limits_{u\in\mu}(c(u)-\varphi(u))$. Повторять этот процесс до получения полного потока φ.

4°. Присвоить целочисленные метки вершинам сети T и знаки «+» или «-» дугам по следующим правилам:

а) входу x_0 присвоить метку 0;

б) если вершине x_i присвоена некоторая метка, а y — еще непомеченная вершина, то вершине $y \in \Gamma x_i$, такой что $\varphi(x_i, y) < c(x_i, y)$ присвоить метку i, а дуге (x_i, y) — знак «+»; вершине $y \in \Gamma^{-1} x_i$, такой что $\varphi(y, x_i) > 0$, присвоить метку i, а дуге (y, x_i) — знак «-». Остальные непомеченные вершины и дуги метки и знака не получают;

в) повторить процесс, описанный в пункте 4°б) до тех пор, пока не прекратится появление новых отмеченных вершин и дуг. Если в результате процесса 4°б) вершина z не получит метки, то поток обладает наибольшей величиной. В противном случае перейти к пункту 5°.

5°. Рассмотреть последовательность отмеченных вершин $\lambda=(z, x_{i_1}, x_{i_2}, ..., x_0)$, у каждой из которых есть метка, равная номеру следующей вершины, и последовательность дуг μ и (не обязательно путь), соединяющих последовательные вершины из λ. Построить новый поток φ':

$$\varphi'(u)=\begin{cases}\varphi(u), \text{если } u \in \bar{\mu},\\ \varphi(u)+1, \text{если } u \in \mu \text{ и имеет знак "+"},\\ \varphi(u)-1, \text{если } u \in \mu \text{ и имеет знак "–"}.\end{cases}$$

Перейти к пункту 4°.

Обоснование алгоритма Форда-Фалкерсона

Заметим, что реализация алгоритма состоит из конечного числа шагов. В самом деле, п. 3° может применяться лишь конечное число раз, так как на каждом шаге величина потока увеличивается, по крайней мере, на единицу. Вместе с тем величина любого потока не может превзойти суммарной пропускной способности дуг, инцидентных выходу сети z, $\sum_{u \in U_z^-} c(u)$.

Процесс присвоения меток конечен в силу того, что каждый раз еще неотмеченные вершины получают метку. И последнее, в п. 5° поток φ' обладает большей величиной, чем φ. Данный вывод вытекает из того, что по определению вершины z и правил п. 4°б) дугам, входящим в z, может быть присвоен только знак «+».

Получаемая по правилу 3° функция φ' — поток. Это непосредственно следует из того, что μ — путь. Также поток строится в соответствии с правилом п. 5°. Для доказательства данного утверждения, рассмотрим три соседние вершины последовательности $\lambda: x_{i_{k-1}}, x_{i_k}, x_{i_{k+1}}$. В силу правила 4°б) возможны только ситуации, представленные на рис. 1. Но в этих ситуациях φ' — поток [6].

Пусть процесс, описанный в п. 4°б), приводит к тому, что вершина z не получает метки. Обозначим через A множество непомеченных вершин. В силу того, что $x_0 \in \bar{A}, z \in A$, множество A определяет разрез сети U_A^-. Каждая дуга $u \in U_A^-$ соединяет помеченную вершину y с непомеченной вершиной $x \in A$. Вершина x может остаться непомеченной при условии, что $\varphi(x,y) = c(y,x)$. Аналогично, на каждой дуге $u \in U_A^-$ (выходящей из A) выполняется равенство $\varphi(u) = 0$, поэтому справедливы соотношения:

$$\varphi_z = \sum_{u \in U_A^-} \varphi(u) - \sum_{u \in U_A^+} \varphi(u) = \sum_{u \in U_A^-} c(u) = c\left(U_A^-\right).$$

В силу леммы это означает, что поток φ обладает наибольшей величиной.

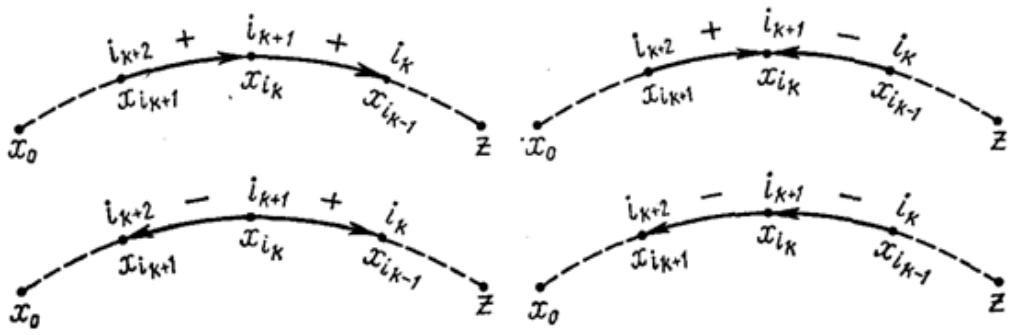

Рисунок 1.1 – Потоки

Проведенное рассуждение совместно с леммой составляют доказательство следующей теоремы [5].

Теорема 1. Для заданной транспортной сети величина наибольшего потока равна наименьшей пропускной способности разрезов, т. е. $\max\limits_{\varphi} \varphi_z = \min\limits_{U_A^-} c\left(U_A^-\right)$.

Пример показан на рисунке 1.2.

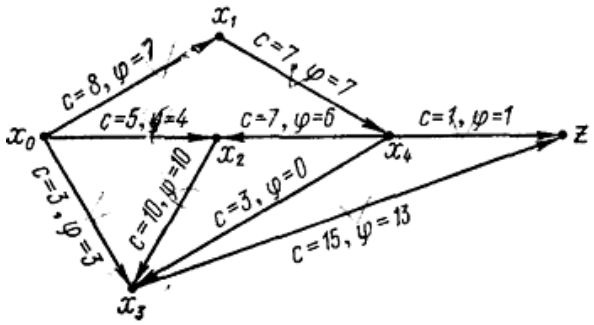

Рисунок 1.2 – Транспортная сеть

Для примера использования алгоритма Форда - Фалкерсона рассмотрим транспортную сеть T и полный поток φ, для которого $\varphi_z = 14$ (рис. 1.2). Применяя правила 4° и 5° алгоритма, можно получить поток с величиной $\varphi_z = 15$.

Задача о назначении на должность (комбинаторная прикладная задача).

Пусть в некотором учреждении имеется 6 вакантных должностей $y_1, y_2, ..., y_6$ и 6 работников $x_1, x_2, ..., x_6$.

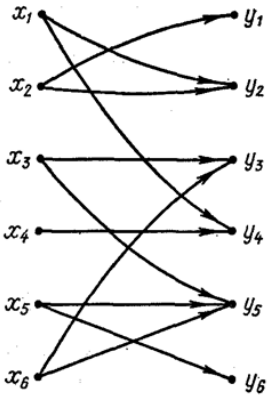

Рисунок 1.3 – Граф

Граф G, изображенный на рис. 1.3, иллюстрирует, какие должности y может занять работник x. Пусть выполнено условие: каждый работник может занимать минимум одну должность и на каждую должность претендует хотя бы один работник [6]. Можно ли назначить работников на должности таким образом, чтобы все шесть должностей заняли работники соответствующей квалификации?

Один из способов решения данной задачи заключается в рассмотрении вспомогательной транспортной сети T. Для ее создания добавим к множеству $X \cup Y$ вершин графа G еще две вершины: вход x_0 и выход z. Соединим x_0 с каждой вершиной x_j дугой пропускной способности $c(x_0, x_j)=1$ и каждую вершину y_i с z дугой пропускной способности $c(x_i, z)=1$. Припишем дугам исходного графа G бесконечные пропускные способности. Получим транспортную сеть T, изображенную на рисунке 1.4

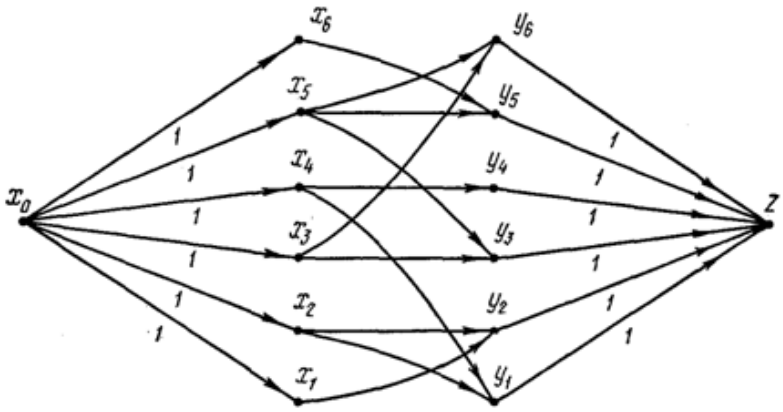

Рисунок 1.4 – Транспортная сеть

Очевидно, что если будет существовать такой поток на сети T, что $\varphi(y_i, z)=1$, то есть и поток, насыщающий выходные дуги (одновременно он будет насыщать и входные дуги (x_0, x_i)), то необходимое назначение будет возможно произвести. Нужно назначить работника x_i на должность y_i в том и только том случае, когда $\varphi(x_i, y_i)=1$.

1.3.2 Алгоритм Диница

Является полиномиальным алгоритмом вычисления максимального потока в транспортной сети.

Этот алгоритм, опубликованный в 1970 г., имел огромное значение. В последующие 10 лет все прорывы в решении задачи о максимальном потоке были основаны на алгоритме Диница [1].

Основная идея метода – алгоритм состоит из фаз, на которых поток увеличивается сразу вдоль всех кратчайших цепей определенной длины. Для этого на i-той фазе строится вспомогательная бесконтурная сеть (layered network). Эта сеть содержит все увеличивающие цепи, длина которых не превышает k_i, где k_i – длина кратчайшего пути из истока в сток. Величина k_i это длина вспомогательной сети [7].

Рассмотрим работу алгоритма на i-той фазе:

Шаг 1. Строим вспомогательную сеть.

Используя поиск в ширину мы движемся из истока в сток по допустимым дугам, добавляя их в S_k и увеличивая k. Дуга u добавляется с $c_k(u) = c(u) – f(u)$. По достижении стока сети, он помечается величиной k и k становится “фиксированной”. Далее продолжается поиск в ширину, но он уже не ведется из вершин v, для которых $q(s, v) \geq k$. Следовательно, вспомогательная сеть будет содержать подсеть исходной. Если поиск в ширину не достиг стока, то работа алгоритма прекращается. Если k больше (k может только увеличиваться) k_i, то мы находимся на i+1 фазе с $k_{i+1} = k$.

Шаг 2. Поиск псевдомаксимального потока.

В бесконтурной сети длины k, которую мы построили, ищем псевдомаксимальный поток. Псевдомаксимальный поток, это такой поток f, для которого не существует увеличивающих цепей длины k, или, по-другому, длины кратчайшего пути. Обнаруженный поток передается в исходную сеть. Возвращаемся к первому шагу [8].

Для построения псевдомаксимального потока используется поиск в ширину (с ограничением на длину пути). Пусть на j-той итерации был найден путь из s в t. Пустим по этому пути поток f_j. Это означает, что, по крайней мере, одна дуга вспомогательной сети насыщена. Удалим все насыщенные дуги. В результате могут возникнуть “тупики”: вершины, из которых не выходит ни одна дуга (кроме стока), вершины, в которые не входит ни одна дуга (кроме источника) и изолированные вершины. Они также должны быть удалены со всеми связанными с ними дугами. Это, в свою очередь, может привести к формированию новых тупиков. Коррекция производится до тех пор, пока в вспомогательной сети не останется ни одного тупика. Изменим пропускные способности оставшихся дуг по формуле $c_k(u) = c_k(u) – f_j(u)$.

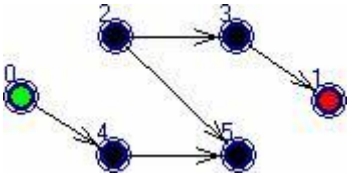

Рисунок 1.5 – Граф

На рисунке 1.5 вершины под номерами 2 и 5, являющиеся тупиками. После их удаления будут последовательно удалены все вершины сети.

Поиск f_j нужно продолжать до тех пор, пока вспомогательная сеть $\neq Ø$. Следовательно, псевдомаксимальный поток $f = \sum f_j$. Псевдомаксимальный поток можно хранить в какой-либо структуре, но, как правило, найденные потоки f_j чаще всего сразу переносятся в исходную сеть S.

Обратим внимание, что как только мы найдём f_i и корректировки сети, можно продолжать поиск в ширину с ближайшей к s не изменившейся дуги найденного пути [9].

После завершения работы Алгоритма исходная сеть будет содержать максимальный поток.

Пример:

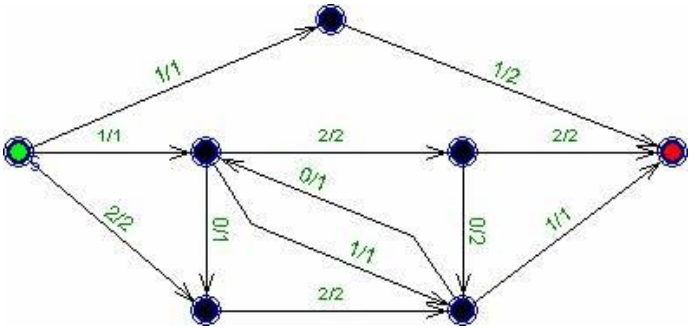

Рисунок 1.6 – Максимальный поток

Сеть с уже найденным с помощью алгоритма Диница максимальным потоком [10].

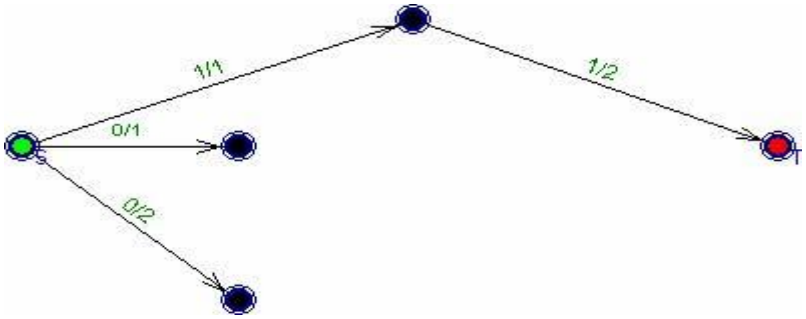

Рисунок 1.7 – Сеть с потоком

Вспомогательная сеть на 1 фазе. $k_1 = 2$.

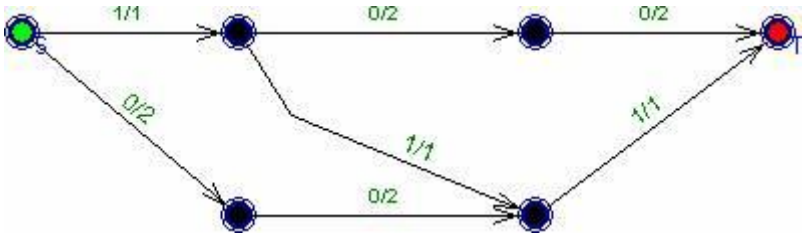

Рисунок 1.8 – Вспомогательная сеть

Вспомогательная сеть на 2 фазе. $k_2 = 3$. На этой сети хорошо видно, что псевдомаксимальный поток обычно не является максимальным.

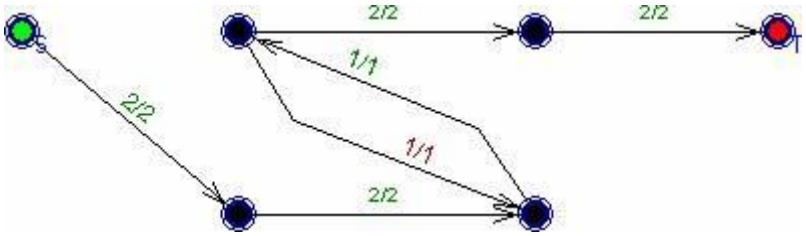

Рисунок 1.9 – Вспомогательная сеть

Вспомогательная сеть на 3 фазе. $k_3 = 5$.

1.3.3 Алгоритм Эдмондса-Карпа

Данный алгоритм был рассмотрен в 1972 Эдмондсом и Карпом, а в 1973 году Диницем. Если, используя этот алгоритм, модифицировать алгоритм кратчайшего возрастания цепей или алгоритм Диница, можно получить оценки быстродействия $O(m^2 Log_n U)$ и $O(mnLog_n U)$ соответственно, где $U = \max(C_{ij})$ [11].

Алгоритм идентичен алгоритму Форда-Фалкерсона, за исключением того, что определяется порядок поиска при поиске пути расширения. Найденный путь должен быть кратчайшим, имеющим доступную емкость. Это можно найти путем поиска по ширине, поскольку мы позволяем ребрам иметь единичную длину.

Возьмем достаточно большое целое число K(так называемый масштаб). Пусть все пропускные способности дуг округляются до ближайшего кратного числа K и формируют сеть CS_k. Затем, по мере увеличения потока в этой сети, он будет увеличиваться на целое число, кратное K. Таким образом, для того, чтобы построить поток мощности M, требуется не более M/K итераций. Как только в сети CS_k будет найден максимальный поток, укажем пропускные способности дуг, выбрав K' и округлим их с недостатком до ближайшего кратного числу K. Где K' < K.

После корректировки сети CS_k и потока f(u), мы получим сеть CS_k' с потоком f'(u). Если K кратно K', то поток, который у нас есть, останется целочисленным [12].

Чтобы поток, полученный на предыдущей итерации снова стал максимальным, его нужно увеличить. Выбирая масштаб, мы все более полно будем применять пропускные способности дуг. В случае с целочисленными пропускными способностями, максимальный поток будет вычислен после его увеличения в масштабе K= 1.

На практике выбирают $K = \log_2 U$. На i-той итерации берется масштаб $2^{(K-i+1)}$. В целочисленном случае для нахождения максимального потока будет достаточно K+1 итераций.

Мы перезагружаем (обнуляем) все потоки. Изначально остаточная сеть совпадает с исходной.

Ищем кратчайший путь от источника до стока в остаточной сети, если такого нет, мы останавливаемся [13].

Пустим через найденный путь (он называется увеличивающим путём или увеличивающей цепью) максимально возможный поток:

На найденном пути в остаточной сети ищем ребро с наименьшей (минимальной) пропускной способностью c_{min}.

Для каждого ребра на найденном пути увеличением поток на c_{min}, а в противоположном ему – уменьшаем на c_{min}.

Модифицируем остаточную сеть. Для всех существующих рёбер на найденном пути, а также для противоположных им рёбер, находим (вычисляем) новую пропускную способность. Если она стала не нулевой, добавляем ребро к остаточной сети, а если она обнулилась, стираем его (ребро).

Вернёмся обратно к шагу 2.

Чтобы найти кратчайший путь на графе, будем использовать поиск по ширине:

Создаём цепочку вершин O. Вначале O состоит из одной вершины s.

Отметим вершину s без предка, как посещенную, а все остальные - непосещённые. Пока цепочка реализуется, выполняем следующие действия:

Вершину u, первую в цепочке, следует удалить.

Для всех (u, v), которые исходят из вершины u, таких, что вершина v ещё не посещена, следует выполнить следующие шаги:

Отметим вершину v как посещённую, с предком u.

Добавим вершину v в конец цепочки, если $v = t$, выйдем из обоих циклов: кратчайший путь найден.

Если цепочка пуста, мы вернем ответ, что пути нет, и остановим алгоритм.

Если нет, переходим от t к s, каждый раз переходя к предку. Мы возвращаем путь в обратном порядке. [14].

Сложность

В процессе работы алгоритм Эдмондса-Карпа будет находить каждый дополняющий путь за время O(V+ E). Можно доказать, что общее число увеличений потока, которое может выполниться с помощью данного алгоритма, составляет O(VE). Таким образом, сложность алгоритма Эдмондса-Карпа равна $O(VE^2)$.

Пример

Для сети из семи узлов, истока A, стока G и пропускных способностей, как показано на рисунке 1.10. Парой f / c обозначен поток этого ребра и его пропускная способность.

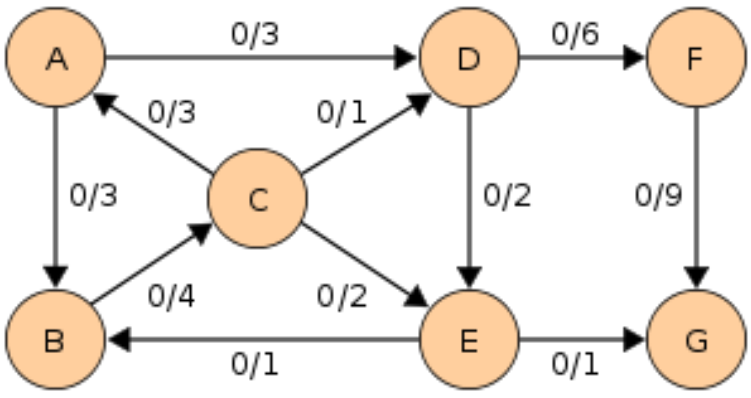

Рисунок 1.10 – Поток

Поиск в ширину

Опишем поиск в ширину на первом шаге.

1. Цепочка состоит из единственной вершины A. Посещена вершина A. Предков нет [15].

2. Цепочка состоит (от начала к концу) из вершин B и D. Посещены вершины A,B,D. Вершины B,D имеют предка A.

3. Цепочка состоит из вершин D и C. Посещены A,B,C,D. Вершины B,D имеют предка A, вершина C – предка B.

4. Цепочка состоит из вершин C,E,F. Посещены A,B,C,D,E,F. Вершины B,D имеют предка A, вершина C - предка B, вершины E,F - предка D.

5. Вершина C удаляется из цепочки: рёбра из неё ведут только в уже посещённые вершины.

6. При обнаружении ребра (E,G) цикл останавливается. В цепочке вершины (F,G). Посещены все вершины. Вершины B,D имеют предка A, вершина C - предка B, вершины E,F - предка D, вершина G - предка E.

7. Идём по предкам: G->E->D->A. Возвращаем пройденный путь в обратном порядке: A->D->E->G.

Заметим, что в цепочку последовательно добавляли вершины, достижимые из A ровно за 1 шаг, ровно за 2шага, ровно за 3 шага. И, кроме того, предок каждой вершины это вершина, достижимая из A ровно на 1 шаг быстрее [16].

1.3.4 Новый алгоритм нахождения максимального потока в многополюсной сети

Разработан новый алгоритм нахождения максимального потока в многополюсной сети, который основан лишь на ее матричном описании и осуществлении тернарных операций над элементами матрицы пропускных способностей дуг, поэтому он не требует графического представления сети.

В силу этого программная реализация разработанного алгоритма является очень простой, и он может быть использован при решении широкого круга проблем, математические модели которых могут быть сформулированы в терминах теории графов [17].

Математические модели широкого круга прикладных проблем могут быть сформулированы в терминах теории графов. В частности, многие такие модели приводят к задаче построения максимального потока в сети с множеством источников и стоков. Известные алгоритмы решения этой задачи существенно основываются на методе расстановки пометок при поиске в сети увеличивающего пути, что предполагает ее графическое представление [18].

Это делает их неудобными для программной реализации. Развита идея осуществления тернарных операций над элементами сети [1], на основе чего разработан алгоритм решения задачи о максимальном потоке с рациональными исходными данными, использующий лишь матричное представление сети (матрица пропускных способностей дуг), лишенный указанных выше недостатков, а увеличение потока на каждой итерации на максимально возможную величину приводит к уменьшению их числа.

Пусть сеть G(V, U) с множеством вершин V = {1, 2, …, n} и множеством дуг U = {(i, j)}, задана матрицей пропускных способностей дуг ее элемент d_{ij} равен пропускной способности дуги (i, j), ведущей из вершины i в вершину j. Естественно полагать если дуга (i, j) не ориентирована, отсутствие дуги (i, j) означает, что $d_{ij} = 0$. Обозначим через S и T множество индексов вершин, которые являются источниками и стоками соответственно, а через − их мощности (a_i − количество потока, которое может выходить из источника − количество потока, которое может принять сток) [20].

Задача заключается в отыскании максимального суммарного потока из всех источников в стоки, при этом для любой вершины сети, не являющейся источником или стоком величины входящих и выходящих потоков, равны.

Таким образом, требуется определить дуговые потоки по дугам, т. е. потоковую матрицу, которая дает решение сформулированной выше задачи. Прежде чем перейти к описанию алгоритма, определим тернарные операции для матрицы, введенные в [2] для вершин сети [19]. Тернарной операцией над матрицей по индексу k называется операция для всех

$$d_{ij} = \max\ d_{ij}, min\ d_{ik}, d_{kj} \quad \text{для всех } i \neq j \neq k. \qquad (1)$$

Рассмотрим вспомогательную матрицу, элементы которой одновременно с выполнением операции (1) изменяются элементы матрицы R по следующему правилу:

$$r_{ij} = \begin{matrix} r_{ij}, \text{если } d_{ij} \geq \min(d_{ik}, d_{kj}) \\ r_{ik}, \text{если } d_{ij} < \min(d_{ik}, d_{kj}) \end{matrix} \qquad (2)$$

Операции (1), (2) являются основой метода построения максимального потока в многополюсной сети [21].

Глава 2 Проектирование системы максимального потока на сетях

2.1 Описание шагов работы алгоритмов

Для ручного способа проверки вычислений используем алгоритм Форда-Фалкерсона.

1. Возьмём поток, изображённый на рисунке 2.1 как начальный допустимый поток. Он имеет величину 3 [25].

2. Присвоим источнику, вершине v_1, метку $(+, \infty)$. Вершина v_1 помечена, но не просмотрена.

3. Просмотрим вершины, смежные с вершиной v_1. Вершине v_2 присвоим метку $(+v_1, 1)$, а вершине v_3 – метку $(+v_1, 1)$,т.к. $\varphi(v_1, v_2) = 2 < 3 = c_1$, $\varphi(v_1, v_3) = 1 < 2 = c_2$. Вершина v_1 помечена и просмотрена, а вершины v_2 и v_3 помечены, но не просмотрены.

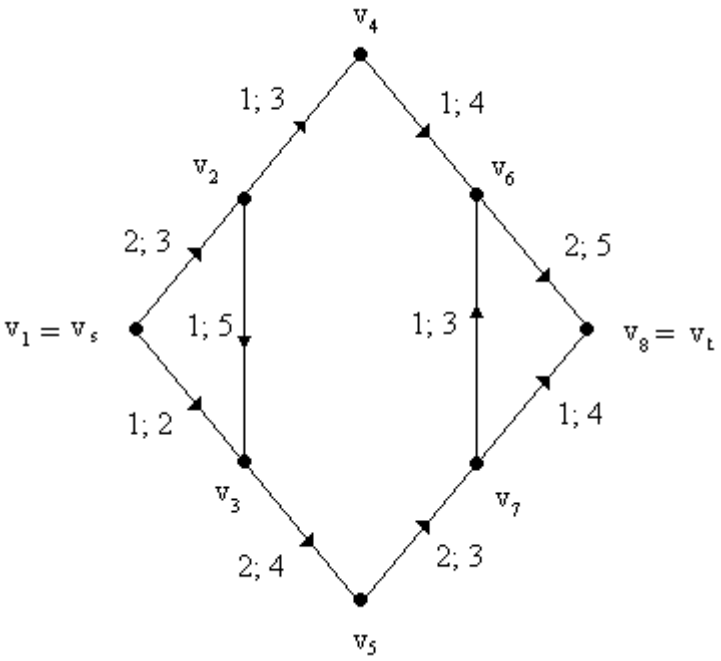

Рисунок 2.1 - Поток величины 3

4. Просмотрим вершины, смежные с вершиной v_2. Из вершин, смежных с вершиной v_2, не помечена только вершина v_4. Вершине v_4 присваиваем метку $(+v_2, 1)$, так как $\varphi(v_2, v_4) = 1 < 3 = c_3$.

5. Просмотрим вершины, смежные с вершиной v_3. Из вершин, смежных с вершиной v_3, не помечена только вершина v_5. Вершине v_5 присваиваем метку $(+v_3, 1)$, так как $\varphi(v_3, v_5) = 2 < 4 = c_4$.

6. Просмотрим вершины, смежные с вершиной v_4. Смежной и непомеченной является вершина v_6. Присваиваем ей метку $(+v_4, 1)$, так как $\varphi(v_4, v_6) = 1 < 4 = c_5$.

7. Просмотрим вершины, смежные с вершиной v_5. Смежной и непомеченной является вершина v_7. Присваиваем ей метку $(+v_5, 1)$, так как $\varphi(v_5, v_7) = 2 < 3 = c_6$.

8. Просмотрим вершины, смежные с вершиной v_6. Смежной и непомеченной является вершина v_8. Присваиваем ей метку $(+v_6, 1)$, так как $\varphi(v_6, v_8) = 2 < 5 = c_7$. Сток помечен. Выполняем операцию Б – увеличение потока [26].

9. Сток имеет метку $(+v_6, 1)$. Значит, увеличиваем поток по дуге (v_6, v_8) на 1.

10. Вершина v_6 имеет метку $(+v_4, 1)$. Значит, увеличиваем поток по дуге (v_4, v_6) на 1.

11. . Вершина v_4 имеет метку $(+v_2, 1)$. Значит, увеличиваем поток по дуге (v_2, v_4) на 1.

12. . Вершина v_2 имеет метку $(+v_1, 1)$. Значит, увеличиваем поток по дуге (v_1, v_2) на 1. Мы получили новый поток величины 4 (рис. 3.9).

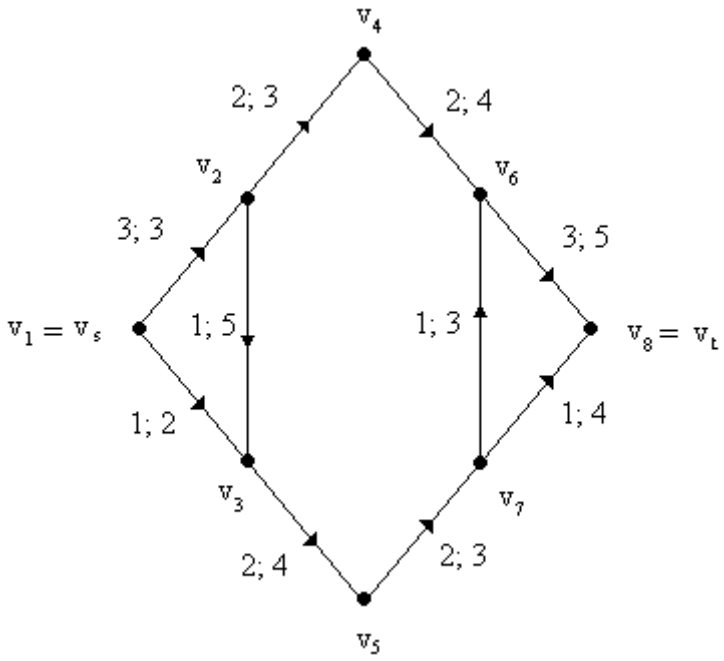

Рисунок 2.2 - Поток величины 4

13. Стираем все метки.

14. Присваиваем вершине v_1 метку (+, ∞).

15. Просмотрим вершины, смежные с вершиной v_1. Вершину v_2 не помечаем, так как $\varphi(v_1, v_2) = 3 = c(e_1)$, а вершине v_3 присваиваем метку $(+v_1, 1)$ [27].

16. Просмотрим вершины, смежные с вершиной v_3. Вершине v_2 присваиваем метку $(-v_3, 1)$, так как $\varphi(v_2, v_3) = 1 > 0$, $l(v) = 1$ и $\min(1, 1) = 1$, а вершине v_5 припишем метку $(+v_3, 1)$, так как $\varphi(v_3, v_5) = 2 < 4 = c_8$.

17. Просмотрим вершины, смежные с вершиной v_4. Вершине v_6 присваиваем метку $(+v_4, 1)$, так как $\varphi(v_4, v_6) = 3 < 5 = c_9$.

18. Просмотрим вершины, смежные с вершиной v_5. Вершине v_7 присваиваем метку $(+v_5, 1)$, так как $\varphi(v_5, v_7) = 2 < 3 = c_{10}$.

19. Просмотрим вершины, смежные с вершиной v_6. Вершине v_8 присваиваем метку $(+v_6, 1)$, так как $\varphi(v_6, v_8) = 3 < 5 = c_{11}$. Сток помечен. Переходим к операции Б – увеличению потока.

20. Сток имеет метку ($+v_6$, 1). Значит, увеличиваем поток по дуге (v_6, v_8) на 1.

21. Вершина v_6 имеет метку ($+v_4$, 1). Значит, увеличиваем поток по дуге (v_4, v_6) на 1.

22. Вершина v_4 имеет метку ($+v_2$, 1). Значит, увеличиваем поток по дуге (v_2, v_4) на 1.

23. Вершина v_2 имеет метку ($-v_3$, 1). Значит, уменьшаем поток по дуге (v_2, v_3) на 1.

24. Вершина v_3 имеет метку ($+v_1$, 1). Значит, увеличиваем поток по дуге (v_1, v_3) на 1. Получили новый поток величины 5 (рис. 3.10).

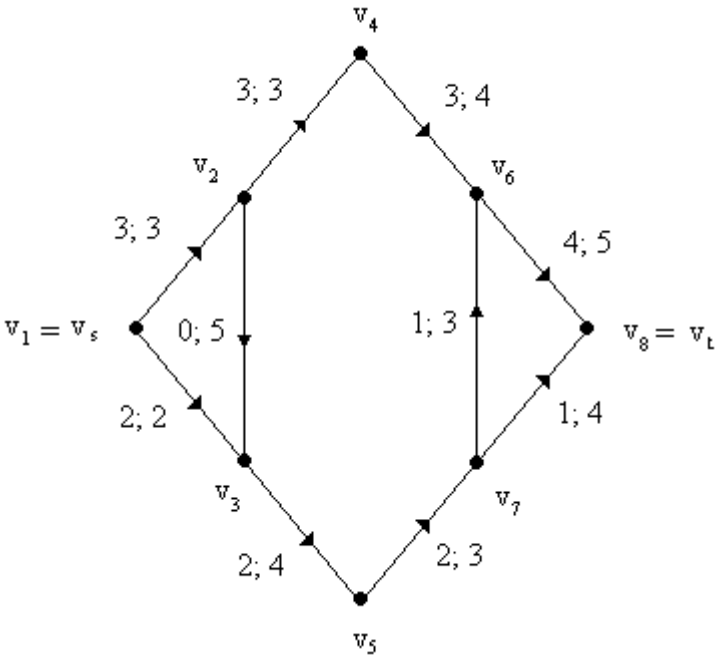

Рисунок 2.3 - Поток величины 5

25. Стираем все метки.

26. Присваиваем вершине v_1 метку ($+, \infty$).

27. Вершины, смежные вершине v_1, нельзя пометить, поскольку дуга (v_1, v_2) насыщена – $\varphi(v_1, v_2) = \varphi(e_1) = c(e_1) = 3$, и дуга ($v_1$, v_3) тоже насыщена – $\varphi(v_1, v_3) = \varphi(e_2) = c(e_2) = 2$. Сток остался непомеченным. Значит, полученный поток максимален [28].

Переходим к поиску максимального потока с помощью программы. Для этого введём матрицу пропускных способностей в соответствующие поля программы. Матрица имеет следующий вид:

$$A = \begin{pmatrix} 0 & 3 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Как мы видим, результат оказался тем же. Мы получили максимальный поток величины 5.

2.2 Алгоритм нахождения максимального потока

Сначала нужно построить полный поток и проверить, можно ли его увеличить. Если нельзя, этот поток – максимальный. Если же его можно увеличить, то будем строить другой полный поток и т.д. Решать задачу будем с помощью метода расстановки меток.

Две основные процедуры (операции алгоритма):

· операция расстановки пометок;

· операция изменения потока.

Рассмотрим первую процедуру. Для каждой вершины данной сети нужно приписать метку следующего вида: (V^+, φ) или (V^-, φ) где $v \in V$, а φ – натуральное число или бесконечность. Возможны три состояния вершины:

1) не помечена;
2) помечена, но не просмотрена;
3) помечена и просмотрена.

Расставлять пометки начнем с источника *S*. Он получит пометку $-;\infty\ ,\varphi\ S\ =\infty$. Источник помечен, но не просмотрен. Остальные вершины не помечены. Чтобы источник S был помечен и просмотрен, надо поместить все вершины, смежные с *S*.

Вершина V_i получит пометку $(S^+,\varphi_{(V_i)})$, где $\varphi_{(V_i)}=min\ \varphi\ S\ ,c\ S,V_i\ -f(S,V_i)$.

Теперь все вершины V_i смежные с S, помечены, но не просмотрены. А вершина S помечена и просмотрена. Начнём просматривать ту из вершин V_i, которая имеет наименьший индекс. Для этого нужно расставить пометки вершинам, смежным с V_i. Если для вершины V_j выполняется следующее условие $f\ V_i,V_j\ <c\ V_i,V_j$, то она получит метку $(V_i{}^+,\varphi\ V_j\ )$, где $\varphi\ V_j\ =min\ \varphi\ V_i\ ,c\ V_i,V_j\ -f\ V_i,V_j$. Если же для вершины V_j выполняется условие $f\ V_i,V_j\ >0$, то V_j получает метку $(V_i{}^-,\varphi\ V_j\ )$, где $\varphi\ V_j\ =min\ \varphi\ V_i\ ,f\ V_i,V_j$. Далее просматриваем следующую вершину, и так до тех пор, пока не пометим сток t или же пока нельзя будет больше пометить ни одной вершины, сток при этом останется не помеченным. Если сток окажется не помеченным, то процесс нахождения максимального потока в сети можно считать законченным, а если сток помечен, то нужно переходить к процедуре 2.

Рассмотрим процедуру изменения потока. Если вершина V_j имеет пометку $(V_i{}^+,\varphi\ V_j\ )$, то $f\ V_i,V_j$ заменяем на $f\ V_i,V_j\ +\varphi\ t$, если же вершина V_j имеет пометку $(V_i{}^-,\varphi\ V_j\ )$, то $f\ V_i,V_j$ заменяем на $f\ V_i,V_j\ -\varphi\ t$.

Переходим к следующей вершине и так до тех пор, пока не достигнем источника S. Здесь изменение потока прекращается. Далее переходим к процедуре 1 и так до тех пор, пока величину потока уже нельзя изменить.

Программа должна находить максимальный поток во введенную в неё транспортной сети.

Глава 3 Автоматизация поиска максимального потока на сетях

3.1 Требования к аппаратной части

Builder C++ 6.0 может работать в среде операционных систем от Windows 98 до Windows XP. Особых требований, по современным меркам, к ресурсам компьютера пакет не предъявляет: процессор должен быть типа Pentium или Celeron с тактовой частотой не ниже 166 МГц (рекомендуется Pentium II 400 МГц), оперативной памяти - 128 Мбайт (рекомендуется 256 Мбайт), достаточное количество свободного дискового пространства (для полной установки версии Enterprise необходимо приблизительно 475 Мбайт) [29].

Спроектируем компьютерную модель базы данных по преподавателям техникума. Что такое «хорошее приложение» зависит от мощности аппаратуры, уровня развития программного обеспечения, вкусов пользователей и, конечно же, постоянно меняется. Мы все же постараемся нарисовать образ хорошего приложения, который будет соответствовать действительности, по крайней мере, лет пять:

– это 32-разрядное приложение для Windows 98 SE или Windows NT, а так же Windows XP и т.д.;

– оно имеет простой, удобный, интуитивно понятный интерфейс со всеми присущими Windows атрибутами: динамическими окнами, кнопками, меню;

– оно управляется как мышью, так и клавиатурой;

– оно отказоустойчиво и корректно обрабатывает любые ошибки пользователя;

– оно черпает информацию из баз данных;

– оно работает быстро, не раздражая пользователя бесконечно медленной прокруткой;

– оно хорошо документировано (как минимум имеет хороший Help);

– оно поддерживает OLE (требования для всех продуктов, желающих получить сертификат от Microsoft);

– оно, возможно, относятся к группе мультимедиа-приложение;

– оно разрабатывается достаточно быстро, чтобы не устареть еще на

– стадии разработки.

3.2 Описание программного кода

Прикладная программа Max_potok написанная на языке C++ в интегрированной среде Builder C++.

В окне программы (Form1: TForm) содержатся такие компоненты, как:

– текстовые поля (Eij: TEdit ; Inij: TEdit (i,j = 1..10));

– компоненты (STy: TStaticText (y = 1..10), OSTx: TStaticText (x = 1..10), VSTz: TStaticText (z = 1..10), OVSTe: TStaticText (e = 1..10), StaticTextij: TStaticText (i = 4, j = 1..4));

– надписи (Label1, Label2, Label3: TLabel);

– панель группировки компонентов (GB1: TGroupBox);

– кнопки (Buttonk: TButton (k = 1..5)).

В программе отслеживается 7 действий (табл. 1).

Таблица 1 – Действия программы

Название действия	Назначение
TButton1.Click	Производит расчёт максимального потока
TButton2.Click	Создаёт таблицы матриц пропускных способностей и потока заданной размерности
TButton3.Click	Считает одну итерацию
TButton4.Click	Выводит метки

TButton5.Click	Начинает итерацию заново
TForm1.Create	Задаёт размер окна программы
TForm1.Close	Совершает затухание формы при её закрытии

Предназначение текстовых полей приведено в таблице 2.

Таблица 2 – Предназначение текстовых полей

Метка поля	Название поля	Предназначение
s=	SSS	Задаёт исток
t=	TTT	Задаёт сток
Вершина	p1	Задаётся вершина, пометки которой будут рассчитываться
Предыдущая вершина	p2	Вывод меток вершин
Знак	p3	Вывод максимального потока сети
δ	p4	
Максимальный поток	max	
Xij	Inij	Задаёт матрицу пропускных способностей сети
Xij	Eij	Вывод матрицы потока в сети

3.3 Руководство программиста

Данная программа осуществляет работу с данными. При использовании компонентов DBEdit, TabSheet, Label, Table, MainMenu, Data Source, RV Project1, RV Table Connektion1. У всех данных компонентов осуществлена взаимосвязь, так как при общем их сочетании мы получаем желательный результат. Без этих компонент работа данного продукта не осуществляется.

Программа состоит из модулей. Каждый модуль соответствует определённой форме и содержит процедуры. Модуль начинает работать сразу же по окончанию работы предыдущего модуля, либо по вызову его пользователем щелчком мыши по соответствующему объекту.

В программе использованы различные свойства компонентов: видимость (visible), доступность (enabled), заголовок (caption) и куча других настроек, необходимых для правильной работы программного продукта.

Таблица 3 – Сравнительный анализ алгоритмов пропускных способностей.

i, j	1	2	3	4	5	6
1	0	30	60	20	0	0
2	50	0	0	0	10	0
3	60	0	0	0	0	40
4	70	0	0	0	40	0
5	0	10	0	20	0	50
6	0	0	10	0	80	0

Сеть в нашем случае выглядит так:

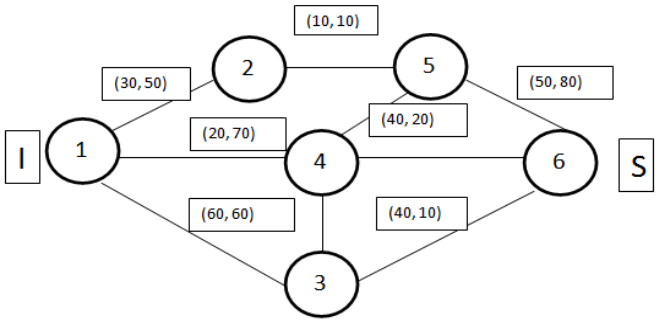

Рисунок 3.6 – Схема вариантов распределения потоков

Необходимо знать следующие правила для решения задачи.

1. Будем считать, что если поток из вершины i к j равен x_{ij}, то противоположный поток равен $(-x_{ij})$.

2. Если поток по дуге x_{ij} меньше его пропускной способности, то есть $x_{ij} < b_{ij}$, то дуга называется ненасыщенной потоком, если же $x_{ij} = b_{ij}$, то дуга называется насыщенной потоком [3, с.146] .

3. Из физического смысла грузопотока следует, что поток по каждой дуге не может превышать ее пропускную способность, т.е. $x_{ij} \le b_{ij}$.

4. Для любой вершины, кроме источника и стока, количество вещества, поступающего в эту вершину, равно количеству вещества, вытекающего из него. Это условие называется условием сохранения потока, в промежуточных вершинах потоки не создаются и не исчезают – отсюда следует, что общее количество вещества, вытекающего из источника, совпадает с общим количеством вещества, поступающего в сток.

Рассмотрим, как организовать какой-нибудь поток на сети.

С этой целью рассмотрим путь 1-2-5-6- это полный путь от источника к стоку. Ребро (2,5) лежащее на этом пути, позволяет пропустить 10 единиц вещества. Следовательно, поток по указанному пути мощностью 10 единиц будет допустимым: $x_{21} + x_{25} = (-x_{12}) + x_{25} = (-10) + 10 = 0$. На пути 1-4-5 можно пропустить 20 единиц вещества (лимитирующим является ребро 1-4). На пути 1-3-6 можно пропустить 40 единиц вещества. В результате потоки по ребрам равны: $x_{12} = 10$, $x_{13} = 40$, $x_{14} = 20$, $x_{25} = 10$, $x_{36} = 40$, $x_{56} = 10+20= 30$, а по остальным ребрам потоки равны нулю. В соответствии с формулой величина сформированного потока равна

$v = x_{12} + x_{13} + x_{14} = x_{36} + x_{56} = 70$ единиц.

3.4 Описание интерфейса программы

Главное окно программы содержит:

– таблицу для введения матрицы пропускных способностей;

– окно для просмотра матрицы максимального потока;

– поля для введения начала и конца транспортной сети [22];

– поле для просмотра величины максимального потока;

– поле для введения номера вершины, пометку которой ми бы хотели увидеть;

– поля для просмотра этих пометок;

– кнопки с алгоритмами (рис. 3.1).

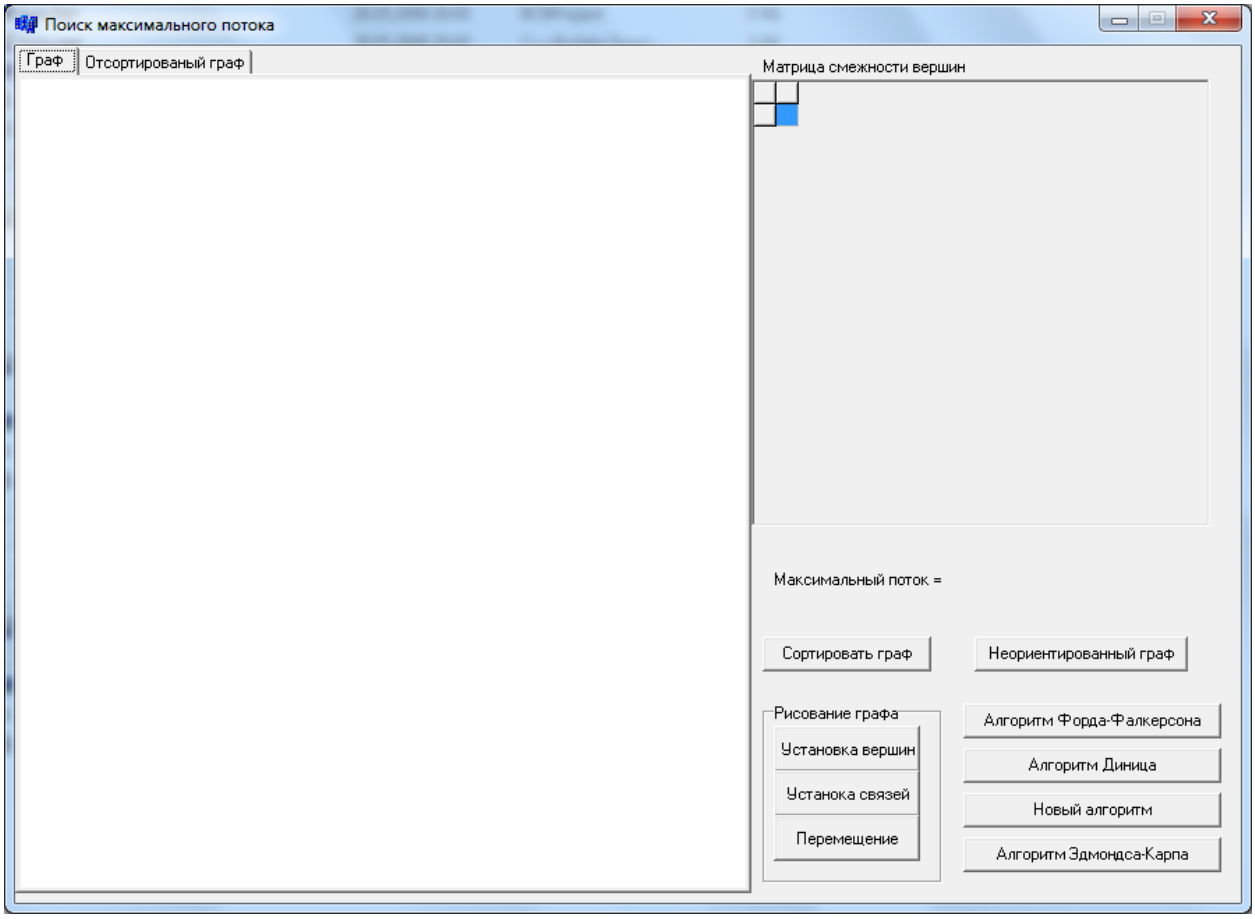

Рисунок 3.1 - Главное окно программы

3.5 Инструкция пользователя

Для запуска программы нужно запустить файл kurs.exe.

Для решения задачи о максимальном потоке в сети необходимо выполнить следующие действия:

1) В зависимости от количества вершин изменятся размеры таблиц, которыми задаются матрицы пропускных способностей и максимального потока сети. На рисунке представлен вид этой таблицы [23].

Нажимаем на кнопку «Установка вершин» и кликаем мышкой на рабочее поле, далее расставляем связи вершин графа, для этого нажимаем на кнопку «Установка связей» и соединяем вершины

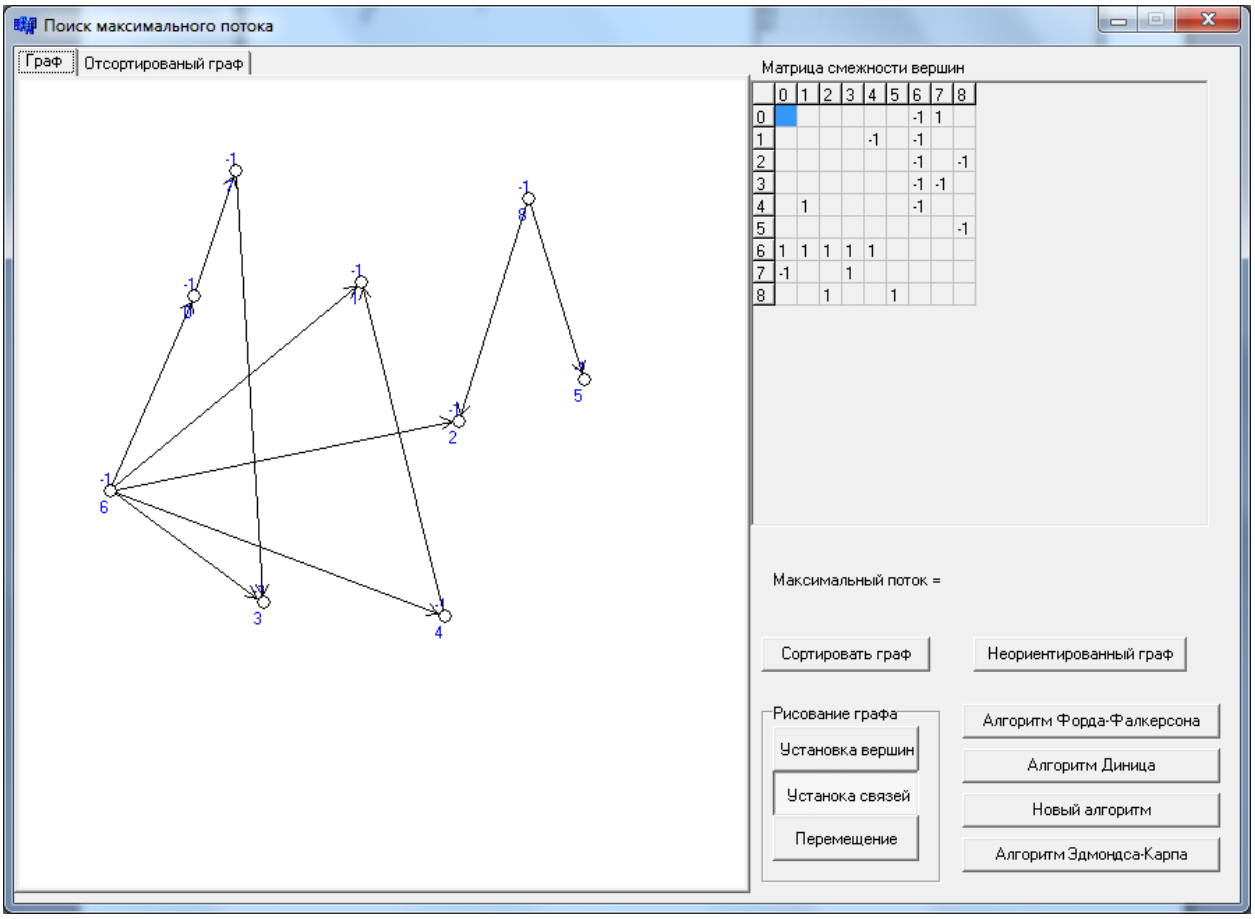

Рисунок 3.2 – Главное окно программы

2) Отменить связь можно нажав правой кнопкой по одной из вершин.

3) При установке связей происходит автоматическое заполнение матрицы смежности вершин.

4) Мы можем самостоятельно изменить веса матрицы.

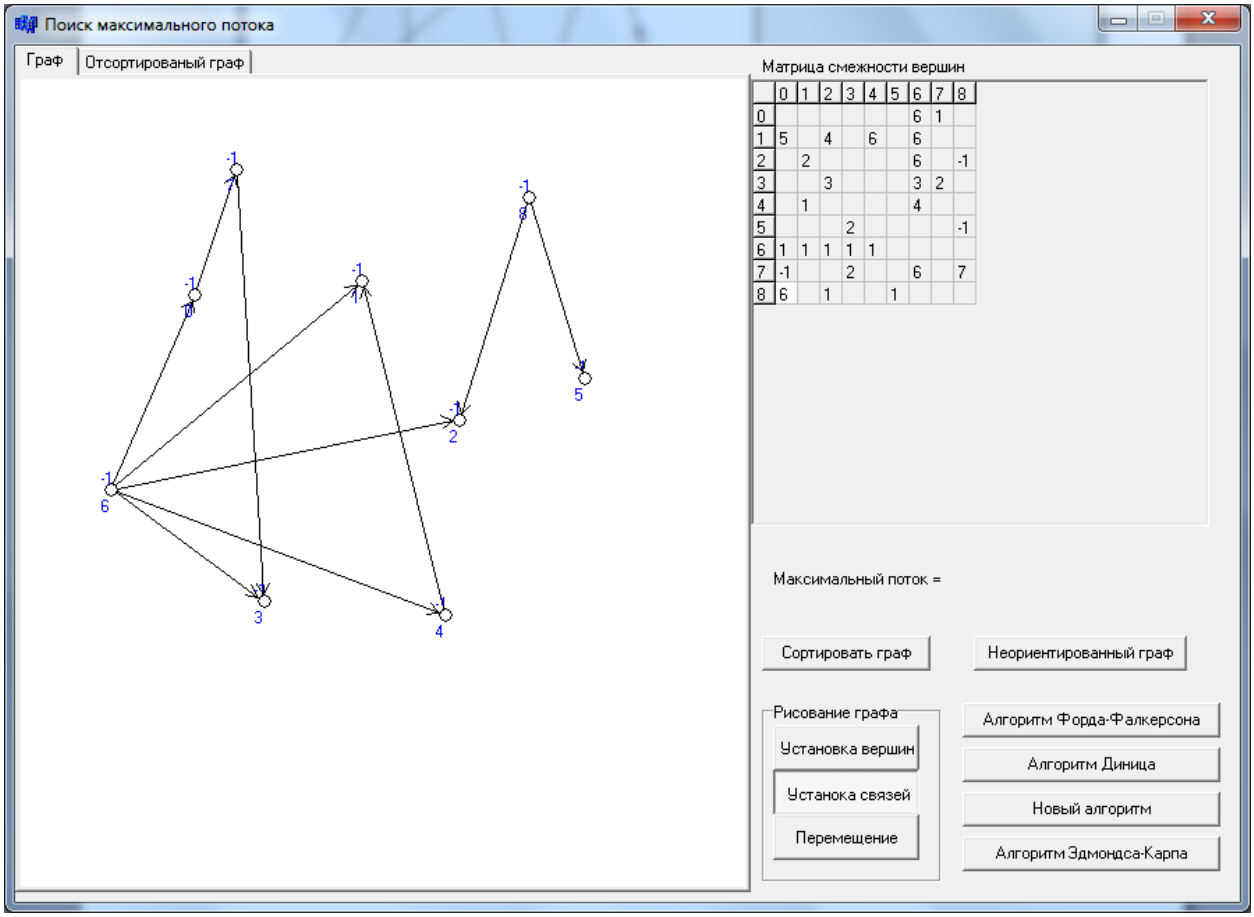

Рисунок 3.3 – Измененная матрица

5) Ввод этих данных – обязательное условие работы программы [24];

6) Рассчитать поток:

– если для выполнения этого этапа использовать кнопку “Алгоритм…”, то в таблице для представления матрицы максимального потока результат отображается немедленно, а в поле “Максимальный поток=” при этом отображается величина максимального потока.

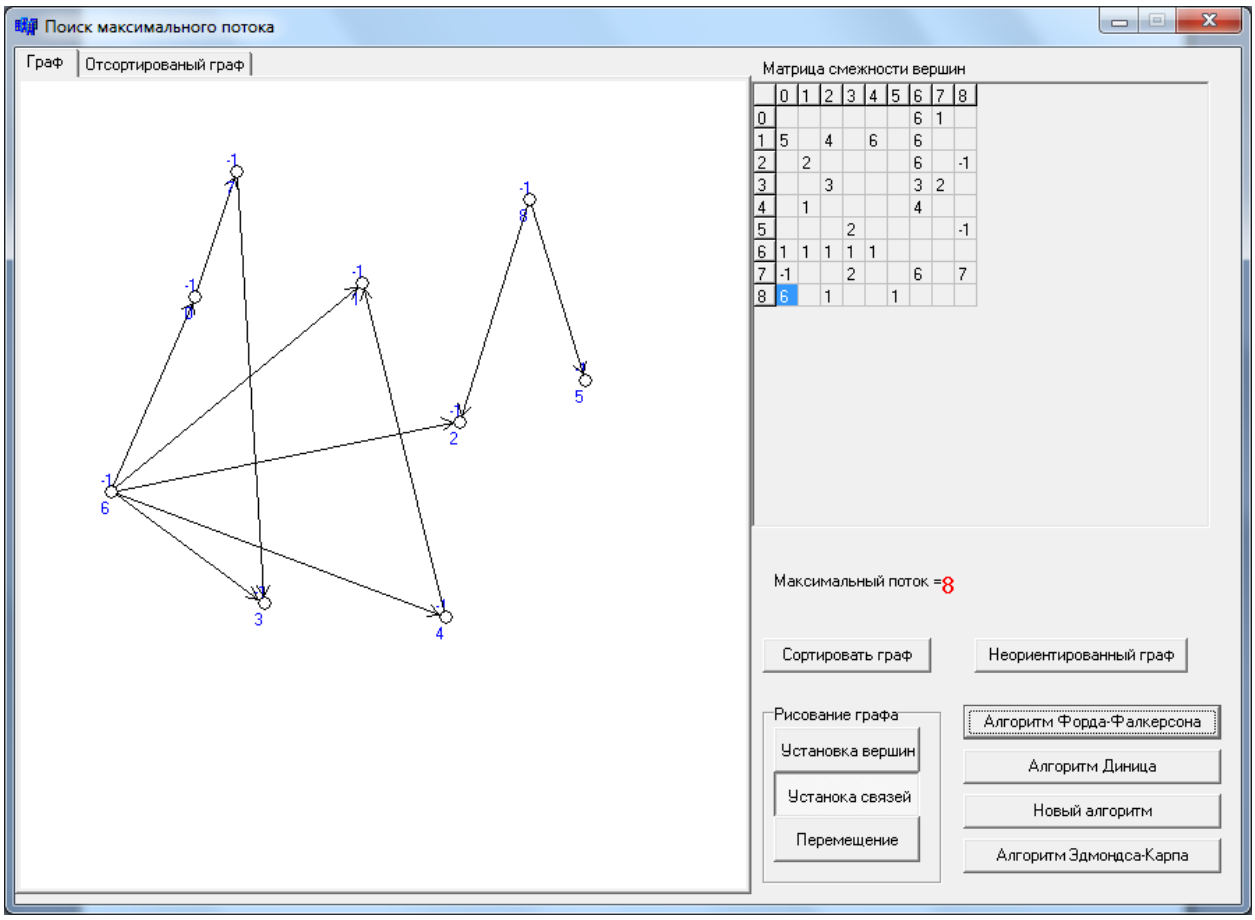

Рисунок 3.4 – Результат

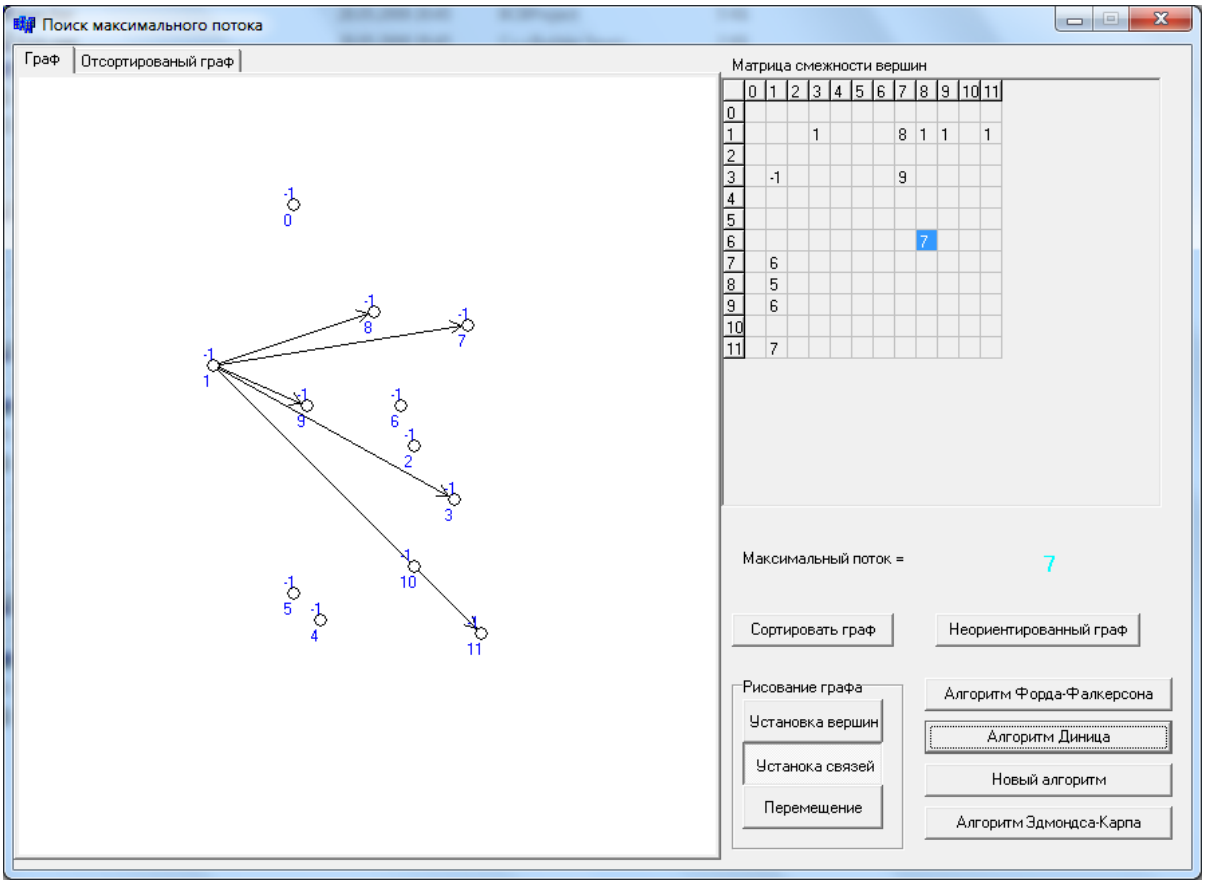

Рисунок 3.5 – Неправильный ввод

Если при решении задачи не соединены вершины, программа выдаст ошибку.

3.6 Сравнительный анализ алгоритмов

В информатике временная сложность алгоритма определяет время работы, используемое алгоритмом, как функции от длины строки, представляющей входные данные. Временная сложность алгоритма обычно выражается с использованием нотации «O» большое, которая исключает коэффициенты и члены меньшего порядка. Если сложность выражена таким способом, говорят об *асимптотическом* описании временной сложности, т.е. при стремлении размера входа к бесконечности. Например, если время, которое нужно алгоритму для выполнения работы, для всех входов длины n не превосходит $5n^3 + 3n$ для некоторого n (большего некоторого n_0), асимптотическая временная сложность равна O (n^3).

Временная сложность оценивается как подсчёт количество элементарных операций, осуществляемых алгоритмом, где элементарная операция занимает для выполнения фиксированное время. Следовательно, полное время выполнения и число элементарных операций, выполненных алгоритмом, отличаются максимум на постоянный множитель.

Так как производительность алгоритма может отличаться при входах одного и того же размера, обычно используется временная сложность наихудшего случая поведения алгоритма, которая обозначается как $\boldsymbol{T(n)}$ и которая определяется как максимальное время, которое требуется для любого входа длины n. Менее часто время измеряется как средняя сложность. Сложность по времени классифицируется природой функции $T(n)$. Например, алгоритм с $T(n) = O(n)$ называется алгоритмом *линейного времени*, а об алгоритме с $T(n) = O(M^n)$ и $m^n = O(T(n))$ для некоторого $M \geq m > 1$ говорят как об *алгоритме с экспоненциальным временем*.

Результаты проведенного сравнительного анализа алгоритмов приведены в таблице 4.

Таблица 4 – Сравнительный анализ алгоритмов решения задачи о максимальном потоке

Алгоритмы	Число операций	Сложность	Описание
Форда-Фалкерсона	4683	O(max| f |)	Найти любой увеличивающий путь. Увеличить поток по всем его рёбрам на минимальную из их остаточных пропускных способностей. Повторять это, пока увеличивающий путь есть. Алгоритм работает только для целых пропускных способностей. Иначе, он может работать бесконечно, не находя правильный ответ.
Алгоритм Диница	9508	$O(V^2E)$	На каждой итерации, используя поиск в ширину, определяем расстояния от источника до всех вершин в остаточной сети. Строим граф, содержащий только такие рёбра остаточной сети, на которых это расстояние растёт на 1. Удаляем из графа все тупиковые вершины с инцидентными им рёбрами, пока все вершины не станут не тупиковыми. На получившемся графе находим кратчайший увеличивающий путь (любой путь из s в t). Исключаем из остаточной сети ребро с минимальной пропускной способностью, снова удаляем тупиковые вершины, и так до тех пор, пока увеличивающий путь есть.
Эдмондса-Карпа	5424	$O(VE^2)$	Вариант алгоритма Форда-Фалкерсона, который каждый раз выбирает кратчайший увеличивающий путь (который можно найти с помощью поиска в ширину).
«Новый алгоритм»	8755	$O(V^3)$ $O(VE \log_n(V^2/E))$ с использованием динамических деревьев	Основан лишь на матричном описании сети и осуществлении тернарных операций над элементами матрицы пропускных способностей дуг, поэтому не требует графического представления сети.

На основе проведенного анализа были выделены следующие достоинства и недостатки рассмотренных алгоритмов:

Алгоритм Форда-Фалкерсона прост в реализации, но работает только для целых пропускных способностей, иначе будет работать бесконечно долго. Алгоритм Диница будет оптимален для использования в задачах с

большим количеством вершин, а алгоритм Эдмондса-Карпа наоборот, будет оптимален для задач с большим количеством рёбер. Главное достоинство нового алгоритма решения задачи о максимальном потоке в том, что он не требует графического представления сети, достаточно матричного описания сети.

ЗАКЛЮЧЕНИЕ

Актуальность задачи о максимальном потоке постоянно возрастает вместе со строительством трубопроводов, новых дорог, роста пользователей Интернета и любых других сетей. Поэтому быстрое и точное её решение крайне необходимо во всех сферах нашей деятельности, где хоть как-то встает вопрос об перемещение чего-либо куда-либо с максимальной рациональностью.

Транспортной сетью называется конечный связный орграф G(V, E) без петель, каждой дуге которого поставлено в соответствие некоторое неотрицательное число, называемое пропускной способностью дуги, и существует:

1) ровно одна вершина, в которую не заходит ни одна дуга, называемая источником или началом сети;
2) ровно одна вершина, из которой не выходит ни одной дуги; эта вершина называется стоком или концом сети.

В данной работе был проведен сравнительный анализ различных алгоритмов решения задачи о максимальном потоке.

В ходе работы были:

1) рассмотрены основные алгоритмы программирования на сетях;
2) разработана система автоматизации поиска максимального потока на сетях;
3) проведен сравнительный анализ реализованных алгоритмов.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Daniel D. Sleator and Robert E. Tarjan (2013). «A data structure for dynamic trees». Journal of Computer and System Sciences 26 (3): 362–391. DOI:10.1016/0022-0000(83)90006-5. ISSN 0022-0000.

2. Daniel D. Sleator and Robert E. Tarjan (2015). «Self-adjusting binary search trees». Journal of the ACM (ACM Press) 32 (3): 652–686. DOI:10.1145/3828.3835. ISSN 0004-5411.

3. Eugene Lawler. 4. Network Flows // Combinatorial Optimization: Networks and Matroids. — Dover, 2011. — P. 109–177. — ISBN 0486414531.

4. Акулич Н.А. Математическое программирование в примерах и задачах. – М.: Высшая школа, 2013.

5. Иозайтис В.С., Львов Ю.А. Экономико-математическое моделирование производственных систем. – М.: Высшая школа, 2010.

6. Миненко С.Н., Гамазина Г.И. Экономико-математическое моделирование производственных систем. – Учебное пособие, МГИУ

7. Таха Хемди А. Введение в исследование операций. 7 издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2016. – 912 с.

8. Таха Хемди А. Введение в исследование операций: В двух книгах. Кн.1.: Пер. с англ. – М.: «Мир», 2015. – 479 с.

9. Вагнер Г. Основы исследования операций. В трех томах. Том 1. Пер. С англ. – М.: Мир, 2014 – 336 с.

10. Аронович А.Б., Афанасьев М.Ю., Суворов Б.П. Сборник задач по исследованию операций. – М. Изд-во МГУ. – 2017. – 256 с.

11. Кормен Томас Х., Лейзерсон Чарльз И., Риверс Рональд Д., Штайн

12. Клиффорд. Алгоритмы: построение и анализ. 2 издание: Пер. С англ. – М.: Издательский дом «Вильямс», 2015. – 1296 с.

13. Басакер Р., Саати Т. Конечные графы и сети. Пер. С англ. – М.: Наука, 2014 – 368 с.

14. Кирсанов М.Н. Графы в Maple. Задачи, алгоритмы, программы. – М.:Издательство ФИЗМАТЛИТ, 2017. – 168 с.

15. Липский В. Комбинаторика для программистов: Пер. с польск. - М:. Мир, 2014. - 213с. ил.

16. Голдберг AV комбинаторной оптимизации лекций для CS363/OR349.

17. Форд Л. Р., Фалкерсон Д. Р. Потоки в сетях. - М:. Мир, 2016.

18. Уилф BS Алгоритмы и сложность. интернет-издания, летом 2014 года. http://www/cis.upenn.edu/wilf.

19. Edmunds. К., Карп RM Теоретические улучшений в эффективности алгоритмических проблем сетевого потока. J. ACM, 2012, 19, с. 248-264.

20. Cormen TH, Leiserson CE, Rivest RL Введение в алгоритмы. 2000 год.

21. Z.Galil , A. Naamad . Сетевого потока и обобщенные сжатия пути. 2013 год.

22. Sleator D. D., Тарьян RE структуру данных для динамических деревьев. J. Comput . Научно системы. , 26:362 -391, 2013.

23. Голдберг А.В., Тарьян RE Новый подход к Задача о максимальном потоке. J. Доц. Comput . Maxa., 35:921-940, 2014.

24. Голдберг А.В., Рао С. Длина функции для потоков вычислений. Технический отчет # 97-055, NEC научно-исследовательский институт, 2017 год.

25. Fong C. O, Рао MR Ускоренная маркировка алгоритмов. Задача о максимальном потоке с приложениями к транспортировке и задач назначения рабочий документ № 7222, Высшая школа менеджмента, У. Рочестера, Рочестер, Нью-Йорк, 2012 год.

26. Лин PM, Леон BJ Повышение эффективности маркировки алгоритмов максимального потока в сетях IEEE Proc Int сиропом схемы и системы, 2014, стр. 162-166.

27. Сринивасан В. , Томпсон GL Ускоренные алгоритмы маркировки и переименование деревьями, с приложениями к проблемам распределения. J. ACM19, 4 (октябрь 2012), 712-726.

28. Джонсон Е Л. Сети и основные решения. Опер . Res. 14 (2016), 619-623.

29. Cheriyan J., Mehlhorn К. Анализ на самом высоком уровне правила отбора в предварительное истечение -PUSH Max-Flow алгоритма (2016).

Приложение

```
//---------------------------------------------------------------------------

//библиотеки

#include <vcl.h>

#pragma hdrstop

#include "myclasses.cpp"

#include "Unit1.h"

#include "Unit2.h"

 //---------------------------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;

const float M_RAD_CONV1=M_PI/180.0;


int PointSize=5;//Радиус точки графа

 int Requrs=0;

int Cycle=0;

//Управляем построением отсортированного графа

int SortLevelDistanceX=8;

int SortLevelDistanceY=8;


//---------------------------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)

   : TForm(Owner)

{

}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
nodes=new TTList<GraphNode*>();
PicSmesh.x=0; PicSmesh.y=0;
PicSmeshMouseDelta.x=0;PicSmeshMouseDelta.y=0;
plane=new Graphics::TBitmap();
plane->Width=PB->Width;
plane->Height=PB->Height;
Modified=true;
}
//---------------------------------------------------------------------------
float TForm1::GetAngle(float x1, float y1, float x2, float y2)
{
float res=0;
if(y1==y2)
  {
  if(x2>=x1)
    res=0;
  if(x2<x1)
    res=M_PI;
  }else
if(x1==x2)//Сразу исключаем, если точки одна над другой.
  {
  if(y2==y1)
    res=0;
```

```
  if(y2>y1)
    res= 90.0*M_RAD_CONV;
  if(y2<y1)
    res= 270.0*M_RAD_CONV;
  }else
  {
  //float dx=x2-x1;
  //float dy=y2-y1;
  //float angle=atan2(y2-y1,x2-x1);
  res=atan2(y2-y1,x2-x1);
  if(res<0)
    {
    res=res*-1;
    res=M_PI-res+M_PI;
    }
  }
return res;
}

void TForm1::DrawArrow(int x1, int y1, int x2, int y2,int radius)
{
int ArrowCone=30;//Угол конуса стрелки
int ArrowConeLength=10;
float angle=GetAngle(x2,y2,x1,y1);
x2=cos(angle)*radius+x2;
y2=sin(angle)*radius+y2;
```

```
x1=cos(angle+M_PI)*radius+x1;

y1=sin(angle+M_PI)*radius+y1;

plane->Canvas->MoveTo(x2,y2);

plane->Canvas->LineTo(x1,y1);

float tangle=angle+(M_PI-ArrowCone*M_RAD_CONV);//Приплюсовываем к углу 180-20 градусов

int tx=cos(tangle)*ArrowConeLength+x1;

int ty=sin(tangle)*ArrowConeLength+y1;

plane->Canvas->LineTo(tx,ty);

tangle=angle+(M_PI+ArrowCone*M_RAD_CONV);//Приплюсовываем к углу 180+20 градусов

tx=cos(tangle)*ArrowConeLength+x1;

ty=sin(tangle)*ArrowConeLength+y1;

plane->Canvas->MoveTo(x1,y1);

plane->Canvas->LineTo(tx,ty);

}

//---------------------------------------------------------------------------

 void __fastcall TForm1::PBMouseDown(TObject *Sender, TMouseButton Button,

   TShiftState Shift, int X, int Y)

{

int nX=X/10;

int nY=Y/10;

bool finded=false;

int i;

if(TabControl1->TabIndex==1)

   {

   PicSmeshMouseDelta.x=X-PicSmesh.x;
```

```
  PicSmeshMouseDelta.y=Y-PicSmesh.y;


  }
for(i=0;i<nodes->Count;i++)
  {
  if((nodes->Get(i)->pos.x==nX)&&(nodes->Get(i)->pos.y==nY))
    {
    finded=true;
    break;
    }
}
if(btnVertex->Down==true)
  {
  if(!finded)
    {
    GraphNode*newnode=new GraphNode(nX,nY);
    nodes->Add(newnode);
    }else
    {
    DeleteNode(i);
    }
  SynchronizeGraphAndGrid();
  Modified=true;
  }else
if(btnLinks->Down==true)
  {
```

```
  if(finded==true)
    {
    if(dragging==true)
      {
      LCV->links->Add(nodes->Get(i));
      }else
      {
      LastClickedVertex.x=nX;
      LastClickedVertex.y=nY;
      LCV=nodes->Get(i);
      NewArrowEnd.x=nX;
      NewArrowEnd.y=nY;
      dragging=true;
      }
    }else
    {
    if(Button==mbRight)
      {
      dragging=false;
      }
    }
  SynchronizeGraphAndGrid();
  Modified=true;
  }else
if(btnMove->Down==true)
  {
```

```
  if(finded==true)
    {
    LCV=nodes->Get(i);
    dragging=true;
    }
  }
PaintPB();
}
//---------------------------------------------------------------------------

bool TForm1::DeleteNode(int vertex)
{
GraphNode*temp=nodes->Get(vertex);
for(int i=0;i<nodes->Count;i++)
  {
  if(i==vertex)continue;
  nodes->Get(i)->DeleteLinksTo(temp);//Удаляем со всех вершин исходящие связи на данную вершину
  }
 delete nodes->Get(vertex);
nodes->Delete(vertex);
Modified=true;
return true;
}
//---------------------------------------------------------------------------
void TForm1::PaintPB()
{
```

```
plane->Canvas->FillRect(Rect(0,0,plane->Width,plane->Height));

//if(TabControl1->TabIndex==0)

//   {

   DrawNodes(TabControl1->TabIndex);

//   }

if((dragging==true)&&(btnLinks->Down==true))

   {


DrawArrow(NewArrowEnd.x*PointSize*2+PointSize,NewArrowEnd.y*PointSize*2+PointSize,
LastClickedVertex.x*PointSize*2+PointSize,LastClickedVertex.y*PointSize*2+PointSize,0);

   }

PB->Canvas->Draw(0,0,plane);

}

//---------------------------------------------------------------------------

void TForm1::DrawNodes(int type)

{

plane->Canvas->Font->Size=9;

int PointDiametr=PointSize*2;

for(int i=0;i<nodes->Count;i++)

   {

   GraphNode*temp=nodes->Get(i);


   int x1; int y1;

   if(type==0)

     {

     x1=temp->pos.x*PointDiametr;

     y1=temp->pos.y*PointDiametr;
```

```
  }else
 if(type==1)
  {
  x1=temp->pos2.x*PointDiametr+PicSmesh.x;
  y1=temp->pos2.y*PointDiametr+PicSmesh.y;
  }
plane->Canvas->Font->Color=clBlue;
plane->Canvas->Brush->Color=(TColor)nodes->Get(i)->rekursnumber;
plane->Canvas->Ellipse(x1,y1,x1+PointDiametr,y1+PointDiametr);
plane->Canvas->Brush->Color=clWhite;
 String text=IntToStr(i);
 SIZE size;
 GetTextExtentPoint32(plane->Canvas->Handle,text.c_str(),text.Length(),&size);
 SetBkMode(plane->Canvas->Handle,TRANSPARENT);
plane->Canvas->TextOutA(x1-size.cx/2,y1+PointDiametr,text);
plane->Canvas->TextOutA(x1-size.cx/2,y1-10,IntToStr(temp->level));
 for(int j=0;j<temp->links->Count;j++)
  {
  GraphNode*tlink=temp->links->Get(j);
  int x2;
  int y2;
  if(type==0)
    {
    x2=tlink->pos.x*PointDiametr+PointSize;
    y2=tlink->pos.y*PointDiametr+PointSize;
    }else
```

```
    if(type==1)
       {
       x2=tlink->pos2.x*PointDiametr+PointSize+PicSmesh.x;
       y2=tlink->pos2.y*PointDiametr+PointSize+PicSmesh.y;
       }
    DrawArrow(x2,y2,x1+PointSize,y1+PointSize,PointSize);
    }
  }
}
void __fastcall TForm1::PBPaint(TObject *Sender)
{
PaintPB();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::PBMouseMove(TObject *Sender, TShiftState Shift,
  int X, int Y)
{
if((TabControl1->TabIndex==1)&&(Shift.Contains(ssRight)))
  {
  PicSmesh.x=X-PicSmeshMouseDelta.x;
  PicSmesh.y=Y-PicSmeshMouseDelta.y;
  PaintPB();
  return;
  }
if(dragging==false)return;
int nX=X/(PointSize*2);
```

```
int nY=Y/(PointSize*2);
if(btnLinks->Down==true)
  {
  NewArrowEnd.x=nX;
  NewArrowEnd.y=nY;
  }else
if(btnMove->Down==true)
  {
  LCV->pos.x=nX;
  LCV->pos.y=nY;
  }
PaintPB();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::PBMouseUp(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
if(btnMove->Down==true)
  {
  dragging=false;
  }
}
//---------------------------------------------------------------------------



void TForm1::SynchronizeGraphAndGrid()
```

```
{
Grid->RowCount=nodes->Count+1;
Grid->ColCount=nodes->Count+1;
for(int i=0;i<nodes->Count;i++)
  {
  Grid->Cells[i+1][0]=IntToStr(i);
  Grid->Cells[0][i+1]=IntToStr(i);
  }
for(int j=0;j<nodes->Count;j++)
    for(int i=0;i<nodes->Count;i++)
      Grid->Cells[i+1][j+1]="";
for(int i=0;i<nodes->Count;i++)
  {
  GraphNode*tn=nodes->Get(i);
  for(int j=0;j<tn->links->Count;j++)//Пройдемся по всем связям данной вершины
    {
    int v=nodes->IndexOf(tn->links->Get(j));//Находим индекс вершины, на которую ссылаемся
    Grid->Cells[v+1][i+1]="1";
    Grid->Cells[i+1][v+1]="-1";
    }
  }
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Clc=0;
for(int i=0;i<nodes->Count;i++)
```

```
  {
  nodes->Get(i)->level=-1;      //Очищаем все данные о уровнях и "цветах" вершин
  nodes->Get(i)->rekursnumber=(int)clWhite;
  }
//В этом цикле мы пройдем по матрице инцеденций, для того, чтобы найти те
//вершины, в которые не входит ни одна стрелка. Если таковых нету, тогда граф циклический,
//и НИЧЕГО ДЕЛАТЬ НЕ БУДЕМ.
int FirstLevelCount=0;
bool DugaFinded=false;
int arrowsInLine=0;
for(int j=0;j<nodes->Count;j++)
  {
  arrowsInLine=0;
  DugaFinded=false;
  for(int i=0;i<nodes->Count;i++)
   {
   if(Grid->Cells[i+1][j+1]=="I")
     {
     DugaFinded=true;
     }
   if(Grid->Cells[i+1][j+1]!="")arrowsInLine++;
   }

  if(arrowsInLine==0)//Обязательно должна быть хотя бы одна связь.
   {
   ShowMessage("Вершина "+IntToStr(j)+" не связана с остальными");
```

```
    break;
    }else
  if(DugaFinded==false)//Нет ни одной входящей дуги - 1 уровень.
    {
    FirstLevelCount++;
    Cycle=0;
    Requrs=0;
    Deep_Deep(nodes->Get(j),0,0);
    }
  }
if(Cycle!=0||Clc!=0)
  {
  ShowMessage("В графе обнаружен цикл. Работа прекращена");
  }
if(FirstLevelCount==0)
  {
  ShowMessage("Не найдено ни одной вершины первого уровня.");
  }

GetSortedGraph();
PaintPB();
Modified=false;
}
//---------------------------------------------------------------------------
```

```
int TForm1::Deep_Deep(GraphNode* node, int nlevel, int ReqNumber)
{
  if((node->level>=nlevel)||(Cycle!=0))return 0;
  if(ReqNumber==node->rekursnumber)
    {
    Cycle=1;
    Clc=1;
    return 0;
    }
  node->level=nlevel;
  node->rekursnumber=ReqNumber;
  for(int i=0;i<node->links->Count;i++)
    {
    Deep_Deep(node->links->Get(i),nlevel+1,ReqNumber);
    ReqNumber=random(10000000);//Выбираем случайное число для идентификации рекурсивной ветки
    }
return 0;
}
//---------------------------------------------------------------------------
bool TForm1::GetSortedGraph()
{
for(int i=0;i<nodes->Count;i++) //Сначала посмотри, отсортирован ли у нас граф
  {
  if(nodes->Get(i)->level==-1)
    return -1;//Не должно быть ни одного -1 уровня.
  }
```

```
int levelx=1;
int levely=14;
int Vert_On_Level=0;
int UpDown=1;//Мы раскидываем по одной, то вверх, то вниз
for(int level=0;level<nodes->Count;level++)
  {
  Vert_On_Level=0;
  levely=14+random(28);
  for(int i=0;i<nodes->Count;i++)
    {
    if(nodes->Get(i)->level==level)
      {
      nodes->Get(i)->pos2.x=levelx;
      nodes->Get(i)->pos2.y=levely+(Vert_On_Level*SortLevelDistanceY*UpDown)+random(20)*UpDown;
      if(UpDown==1)
      Vert_On_Level++;

      UpDown*=-1;
      }
    }
  levelx+=SortLevelDistanceX;
  }
return true;
}
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
```

```
PicSmesh.x=0;
PicSmesh.y=0;
PicSmeshMouseDelta.x=0;
PicSmeshMouseDelta.y=0;
dragging=false;
PaintPB();
if(TabControl1->TabIndex==1)
  {
 // Button1->Enabled=false;
  btnLinks->Down=false;
  btnLinks->Enabled=false;
  btnMove->Down=false;
  btnMove->Enabled=false;
  btnVertex->Down=false;
  btnVertex->Enabled=false;
  }else
  {
  Button1->Enabled=true;
  btnLinks->Down=false;
  btnLinks->Enabled=true;
  btnMove->Down=false;
  btnMove->Enabled=true;
  btnVertex->Down=false;
  btnVertex->Enabled=true;
  }
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::TabControl1Changing(TObject *Sender,
  bool &AllowChange)
{
if(Modified==true)AllowChange=false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form2->CreateGraph(nodes);
Form2->ShowModal();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button3Click(TObject *Sender)
{
int Kolonka=0;
  for(int i=1;i<Grid->RowCount;i++)
   Kolonka += Grid->Cells[Grid->ColCount-1][i].ToIntDef(0);
  int Stroka=0;
  for(int i=1;i<Grid->ColCount;i++)
   Stroka += Grid->Cells[i][Grid->RowCount-1].ToIntDef(0);

 Label3->Caption=FloatToStr(Stroka);
```

```
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button4Click(TObject *Sender)
{
  int Kolonka=0;
  for(int i=1;i<Grid->RowCount;i++)
    Kolonka += Grid->Cells[Grid->ColCount-1][i].ToIntDef(0);
  int Stroka=0;
  for(int i=1;i<Grid->ColCount;i++)
    Stroka += Grid->Cells[i][Grid->RowCount-1].ToIntDef(0);

 Label4->Caption=FloatToStr(Stroka);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
 int Kolonka=0;
  for(int i=1;i<Grid->RowCount;i++)
    Kolonka += Grid->Cells[Grid->ColCount-1][i].ToIntDef(0);
  int Stroka=0;
  for(int i=1;i<Grid->ColCount;i++)
    Stroka += Grid->Cells[i][Grid->RowCount-1].ToIntDef(0);
```

```
 Label5->Caption=FloatToStr(Stroka);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
  int Kolonka=0;
  for(int i=1;i<Grid->RowCount;i++)
    Kolonka += Grid->Cells[Grid->ColCount-1][i].ToIntDef(0);
  int Stroka=0;
  for(int i=1;i<Grid->ColCount;i++)
    Stroka += Grid->Cells[i][Grid->RowCount-1].ToIntDef(0);

 Label6->Caption=FloatToStr(Stroka);
}
//---------------------------------------------------------------------------
```