Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«ПРОГРАММНОЕ СРЕДСТВО ДЛЯ НАХОЖДЕНИЯ

МАКСИМАЛЬНОГО ПОТОКА В СЕТИ»

БГУИР КП 1-40 01 01  016  ПЗ

Студент: гр. 951005 Доведько Д.Ю.

Руководитель: асс. Болтак С.В.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

Лапицкая Н.В. 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Доведько Диане Юнесовне

1. Тема работы Программное средство для нахождения максимального потока сети

2. Срок сдачи студентом законченной работы 27.12.2020

3. Исходные данные к работе среда разработки Visual Studio, файлы с текстом программы на языке С/С++.

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение.*

*1. Анализ литературных источников и постановка задач;*

*2. Описание алгоритма Эдмондса-Карпа;*

*3. Разработка программного средства;*

*4. Технические приемов программирования;*

*5. Тестирование;*

*6. Руководство пользователя;*

*Заключение, список литературы, ведомость, приложения.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема программы

6. Консультант по курсовому проекту Болтак С.В.

7. Дата выдачи задания 11.09.2020 г.––   –

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1, введение к 06.10.2020 – 10 % готовности работы;

разделы 2 к 27.10.2020 – 30 % готовности работы;

разделы 3,4 к 17.11.2020 – 60 % готовности работы;

раздел 5, 6 к 08.12.2020 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 25.12.2020 – 100 % готовности работы.

Защита курсового проекта с 25.12 2020 г. по 06.01 2021 г.

РУКОВОДИТЕЛЬ С.В. Болтак

(подпись)

Задание принял к исполнению –––\_\_\_\_––     11.09.2020 г.

(дата и подпись студента)

**СОДЕРЖАНИЕ**

Введение .................................................................................................. 4

1 Анализ литературных источников и постановка задач................... 5

1.1 Перечисление исследованных литературных источников…... 5

1.2 Общая постановка задачи............................................................ 5

1.3 Постановка вопросов, необходимых для определения

детализированных задач...................................................................... 5

1.4 Постановка детализированных задач….….……...……….......... 7

2 Описание алгоритма Эдмондса-Карпа............................................... 8

2.1 Словесное описание алгоритма Эдмондса-Карпа..................... 8

2.2 Схема алгоритма Форда-Фалкерсона......................................... 17

2.3 Оценка сложности и эффективности алгоритма........................ 17

3 Разработка программного средства.................................................... 19

3.1 Логическое разделение структуры программы.......................... 19

3.2 Файлы EdmondKarpAlgroithm.h и EdmondKarpAlgorithm.cpp.. 19

3.3 Тестирование EdmondsKarpAlgorithm.cpp................................. 20

3.4 Файл Source.cpp............................................................................. 21

3.5 Файл resource.h.............................................................................. 34

4 Технические приёмы программирования…………………………… 35

5 Тестирование……………………......................................................... 36

6 Руководство пользователя…………………………………………… 48

Заключение……....................................................................................... 56

Список использованных источников...................................................... 57

Приложение. Исходный код программы…………………………...…. 58

**ВВЕДЕНИЕ**

Многие вычислительные приложения и программные средства оперируют не только множеством некоторых объектов, но и множеством связей между парами этих объектов. Наличие связей между парами этих объектов довольно часто приводит к задачам о нахождении кратчайшего пути от одного объекта к другому, о нахождении множества всех достигаемых объектов из заданного и о нахождении наиболее выгодного пути.

Для моделирования подобных задач и поиска их решения используются графы – абстрактные модели в математике, объединяющие в себе множество вершин и множество ребер.

Теория графов – это раздел дискретной математики, изучающий свойства графов. Основоположником теории графов считается выдающийся математик и физик Леонард Эйлер, в 1736 году сформулировавший и решивший задачу о семи кёнигсбергских мостах, которая впоследствии стала классической задачей теории графов.

За несколько столетий были математически доказаны многие важные и полезные свойства графов, однако некоторые наиболее сложные задачи в теории графов так и остались нерешенными. Несмотря на то, что возраст самых фундаментальных алгоритмов на графах насчитывает несколько столетий, много алгоритмов, имеющих практический интерес, было изобретено именно в последние два десятилетия.

Теория графов находит широкое практическое применение в различных областях науки и техники и промышленных отраслях: химии, экономике, логистике, физике, информатике, программировании, схемотехнике и др.

Графы могут применяться для моделирования компьютерных сетей, поставок товаров, транзакций, совершаемых по сети Интернет, движения городского транспорта, а также электрических цепей с током. Особый интерес представляет собой задача о нахождении максимального потока в сети, для решения которой существуют алгоритмы Форда-Фалкерсона, Эдмондса-Карпа и Диница.

Решение задачи о нахождении максимального потока позволило решить множество других задач в теории графов, такие как нахождение максимального разреза и всех паросочетаний графа, а также ряд перечислительных задач.

Цель курсового проектирования – создание программного средства для нахождения максимального потока сети алгоритмом Эдмондса-Карпа с использованием принципов процедурного программирования, принципов объективности, целенаправленности и системности научного исследования.

**1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И**

**ПОСТАНОВКА ЗАДАЧ**

**1.1 Перечисление исследованных литературных источников**

Для разработки программного средства для нахождения максимального потока сети были исследованы различные литературные источники.

Были исследованы общий принцип работы алгоритма Эдмондса-Карпа, смысл пропускной способности и потока, минимальной пропускной способности, а также теорема Форда-Фалкерсона [1].

Были изучены доказательства теоремы Форда-Фалкерсона, а также доказательства оценки сложности алгоритма [1][2][3].

Были более детально и глубоко изучены принципы работы алгоритма Эдмондса-Карпа, а также практические области применения данного алгоритма. Было проведено ознакомление с другими алгоритмами нахождения максимального потока сети [3].

Поскольку в качестве языка программирования для написания программного средства можно было выбрать язык Си либо С++, было проведено ознакомление со всеми возможностями данных языков и с их обширной библиотекой готовых функций (классов) [4][5][6][7], а также ознакомление с готовыми средствами языков С/С++ (или написанными на этих языках), позволяющими создавать GUI (Graphical User Interface – графический интерфейс пользователя) [7][8][9].

**1.2 Общая постановка задачи**

Общая задача: написать приложение, получающее в качестве входных данных некоторую сеть и после выполнения работы возвращающее в качестве результата сеть с найденными пропускными способностями (с выводом максимального потока сети).

**1.3 Постановка вопросов, необходимых для определения**

**детализированных задач**

По завершении анализа литературных источников для рассмотрения были выдвинуты следующие вопросы:

1. Каким образом удостовериться в корректности работы реализованного алгоритма Эдмондса-Карпа?
2. Каким образом хранить информацию о вводимой пользователем сети? Таблицей или списком смежности? Привлечь ли другие способы хранения графов?
3. Каким образом найти кратчайший путь в остаточной сети?
4. Какой использовать инструмент для создания GUI?

Посредством метода аналитического сравнения были сделаны следующие выводы:

1. Реализуемое программное средство должно всегда давать правильный ответ на поставленную перед ним задачу, а именно – находить максимальный поток сети, вводимой пользователем. Всё возможное многообразие сетей перебрать вручную для тестирования программы достаточно сложно, поэтому во время разработки программного средства остро стоял вопрос о должном и качественном тестировании реализованного алгоритма.

Было принято решение при реализации алгоритма Эдмондса-

Карпа в первую очередь отдать код с реализацией на тестирование проверяющей системе сайта *informatics.mccme.ru* [10]*.* Данный сайт обладает обширным архивом олимпиадных задач по программированию и надежной тестирующей системой для проверки решений, высылаемых пользователями. Задача о нахождении максимального потока сети имеет номер 2783 в общем архиве задач.

1. Существует множество способов представления графов в памяти компьютера. Использование того или иного способа имеет так свои достоинства и недостатки, причем способ представления графа напрямую может влиять не только на производительность программного средства, но и еще на сложность реализации того или иного алгоритма, применяемого на графе для решения поставленной задачи.

После оценки ограничений на входные данные, которые будут

поступать в программное средство, было принято решение хранить информацию о сети в программе при помощи таблицы смежности. Это в первую очередь связано с тем, что программное средство будет искать максимальный поток для сетей с количеством вершин, не превышающих 20, а также с количеством дуг, не превышающим 100. Использование таблицы смежности заметно повысит эффективность используемой памяти в случае достаточно плотных сетей с большим числом дуг. Кроме того, в случае с использованием таблицы смежности, реализация алгоритма Эдмондса-Карпа становится достаточно простой.

1. В ходе работы алгоритма Эдмондса-Карпа одним из наиважнейших этапов является нахождение наикратчайшего пути в остаточной сети. Существует множество алгоритмов для нахождения наикратчайшего пути в графе. Приведем сравнение некоторых из них.

Алгоритм Флойда – O(N^3).

Алгоритм Дейкстры – O(N^2).

Алгоритм Беллмана-Форда – O(N \* M).

Предпочтение было отдано алгоритму Дейкстры, поскольку он обладает большей эффективностью по сравнению с другими алгоритмами (алгоритм Флойда проигрывает из-за своей кубической сложности, в то время как эффективность алгоритма Беллмана-Форда зависит не только от количества вершин, но и от количества дуг, которых может быть намного больше, чем вершин).

1. Из средств создания GUI при помощи языков C/C++ были изучены и сравнены между собой следующие представители: WinAPI, Qt, C++ Builder и Microsoft Foundation Classes. Предпочтение было отдано WinAPI (Windows Application Programming Interface), поскольку именно он как нельзя кстати подходит для наших целей: Qt, C++ Builder и MFC представляют собой довольно мощные инструменты для быстрого создания высококачественного GUI, поскольку все они содержат практически готовые решения в виде кнопок, таблиц, встроенных веб-обозревателей, звуковых воспроизводителей и т.д. Но эти готовые решения не позволят нам в полной мере осуществить ввод пользователем сети в наше программное средство, поскольку они изначально не предназначены для этого. В свою очередь, WinAPI позволяет нам целиком определить цикл обработки сообщений каждой кнопки и каждого клика пользователя, что предоставляет ту свободу и гибкость, так нам необходимую для создания программного средства.

**1.4 Постановка детализированных задач**

Задачи:

* используя таблицу смежности и алгоритм Дейкстры, реализовать алгоритм Эдмондса-Карпа, находящий максимальный поток вводимой через консоль сети;
* отправить код с реализацией на тестирование проверяющей системе, в случае успешного прохождения всех тестов, продолжить работу над созданием программного средства;
* создать структуры EDGE и VERTEX для хранения вводимых пользователем сети посредством GUI и корректного их преобразования для передачи необходимых данных алгоритму Эдмондса-Карпа;
* используя WinAPI, создать удобный графический интерфейс пользователя с перерисовкой всех элементов управления в реальном времени.

**2 ОПИСАНИЕ АЛГОРИТМА ЭДМОНДСА-КАРПА**

**2.1 Словесное описание алгоритма Эдмондса-Карпа**

Перед тем как перейти к непосредственному описанию алгоритма Эдмондса-Карпа необходимо дать определение сети.

Сеть – это ориентированный граф, имеющий две особые вершины – сток и исток, такие, что любая другая вершина обязательно лежит на пути из истока в сток.

Ориентированным графом называются граф, рёбрам которого присвоено направление.

Всем ребрам (дугам) сети назначена некоторая неотрицательная пропускная способность. При помощи подобной сети можно моделировать транспортную сеть, поставки, трубопроводные и электрические сети. Пример обесточенной сети можно увидеть на рисунке 2.1.

Рассмотрим важное свойство сети – в вершинах сети, не являющимися стоком или истоком, применяемый нами в некоторой ситуации тип объекта (машины, поставки, вода, ток и т.д.) не накапливается, т. е. действует правило аналогичное первому правилу Кирхгофа («сколько входит в вершину, столько и выходит»). Проверим выполнение этого свойства на сети, представленной на рисунке 2.2.

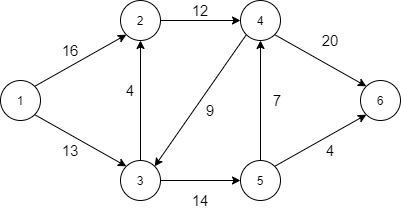


Рисунок 2.1

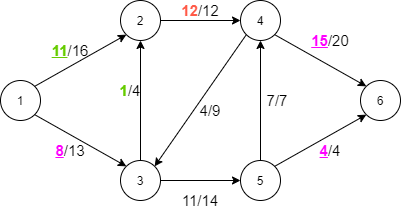


Рисунок 2.2

На рисунке 2.1 у каждого ребра указана его пропускная способность.

На рисунке 2.2 в вершину с номером 2 поступает 12 единиц потока (выделены салатовым), все эти 12 единиц поток перенаправляются в вершину с номером 4 (выделены красным). В свою очередь, в вершину с номером 4 поступает 19 единиц потока от вершин с номерами 2 и 5. Все 19 единиц поток разделяются: 15 единиц потока перенаправляется в вершину с номером 6, другие 4 единицы перенаправляются в вершину с номером 3. Соблюдение вышеупомянутого свойства сети можно убедиться глобальнее: из вершины-истока выходят 11 + 8 = 19 единиц потока (выделены салатовым и розовым), все эти 15 + 4 = 19 единиц потока (выделены розовым) поступают в вершину-сток. Соответственно, величина потока сети равна 19.

Таким образом, алгоритм Эдмондса-Карпа решает следующую задачу – происходит поиск такого способа распределить потоки по ребрам сети, чтобы в сток попало как можно больше потока (воды, электричества и т.д.) и чтобы при этом нигде поток не превышал заданную пропускную способность (в противном случае это чревато «взрывом трубы» или «повреждением проводника»). Теперь перейдем к более формальной формулировке понятия «сеть», выставляемых требований к ней, а также задачи о максимальном потоке.

* **Сеть G = (V, E)** – это ориентированный граф, в котором каждое ребро **(u, v)  E** имеет неотрицательную пропускную способность **c(u, v) ≥ 0**
* Если **E** содержит ребро **(u, v)**, то не существует ребра **(v, u)** в обратном направлении (необходимость этого свойства станет очевидна позже).
* Если **(u, v) E**, то **с(u, v) = 0**. Сеть не содержит петель.
* Выделяются две вершины – **исток s** и **сток t**. Всякая другая вершина лежит на пути от **s** к **t**.
* **Поток в G** – это **функция** , удовлетворяющая двум свойствам:

1. Ограничение **пропускной способности**:

Сохранение **потока**:

* Если **(u, v) E**, то **f(u, v) = 0**.
* **Величина потока** – это сумма потоков из источника.
* **Задача о максимальном потоке** – найти такое **f**, чтобы **|f|** была максимальна.

Рассмотрим граф, представленный на рисунке 2.3. В этом графе есть противоположно направленные ребра (2, 3) и (3, 2). Наличие таких ребер запрещено согласно нашему определению сети. В таком случае создается вершина ***v’***, через которую будет перенаправлено одно из противоположно направленных ребер. Граф, представленный на рисунке 2.4 уже является сетью. Математически доказано, что данные графы эквивалентны друг другу в решаемой нами задаче.

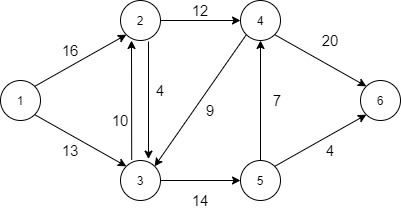


Рисунок 2.3

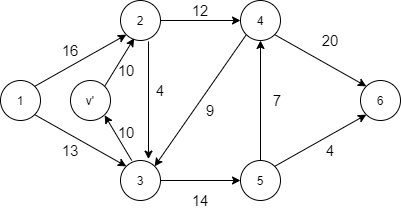


Рисунок 2.4

Представляют интерес сети, имеющие несколько истоков и стоков (рисунок 2.5). Каким образом их свести к сетям с одним стоком и одним истоком? Достаточно добавить вершины s и t, причем ребра, исходящие из s в t имеют пропускную способность + (рисунок 2.6).

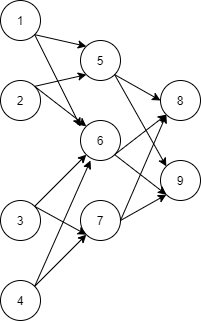


Рисунок 2.5

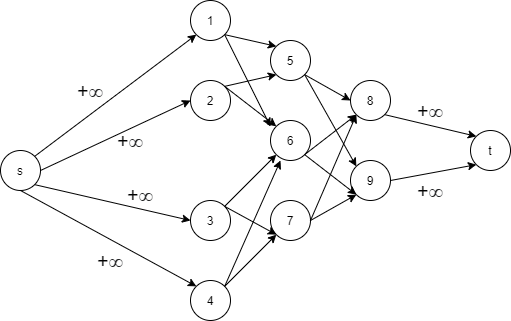


Рисунок 2.6

Во время работы любого алгоритма поиска максимального потока необходимо точно знать не только сколько в данный момент через конкретное ребро течет потока, но также и насколько мы можем увеличить «напор» этого потока в данном ребре. Кроме того, всегда необходимо знать, сколько потока может быть перенаправлено обратно, если найден более выгодный способ распределения потоков. Для этого вводятся понятия **остаточной сети** и **остаточной пропускной способности**.

Пусть дана **сеть G = (V, E)** с **потоком f** и **пропускной способностью c**. **Остаточная пропускная способность cf(u, v)** определяется следующим образом:

В первом случае остаточная пропускная способность показывает, насколько мы можем увеличить напор, а во втором – сколько единиц потока мы можем направить обратно.

Введем понятие **множества остаточных ребер**:

Стоит отметить, что может содержать ребра, которых нет в Причем это множество () будет содержать противоположно направленные ребра (то есть у каждой пары вершин (если они соединены) в остаточной сети будет не более одной пары противоположно направленных ребер). Чтобы добиться этого, мы избавлялись от противоположно направленных ребер в исходном графе.

**Остаточной сетью** будем называть граф (на рисунке 2.8 представлена остаточная сеть для сети, изображенной на рисунке 2.7).

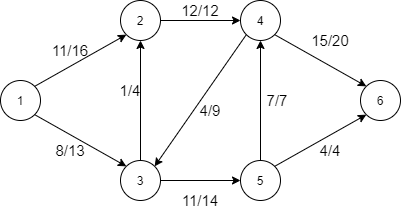


Рисунок 2.7

Теперь введем понятие **дополняющего пути и остаточной пропускной способности**. **Дополняющий путь p** – это любой путь из s в t остаточной сети. На рисунке 2.9 выделен красным дополняющий путь p.

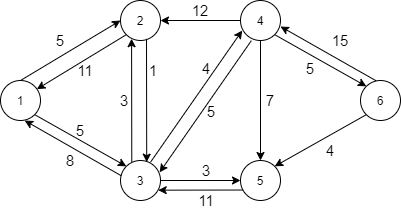


Рисунок 2.8

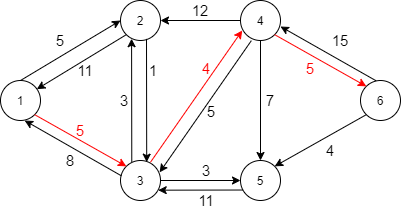


Рисунок 2.9

**Остаточная пропускная способность** – это то максимальное число, на которое мы можем увеличить поток из дополняющего пути. Для выбранного p она равна 4. Если мы сделаем попытку увеличить поток на 5, то нарушится баланс (ребро 3-4 станет равно -1). Формальное определение:

Взяв в качестве примера остаточную сеть на рисунке 2.9, увеличим поток вдоль красного пути на 4 единицы, получим сеть и остаточную сеть, представленные на рисунках 2.10 и 2.11 соответственно.

Как можно заметить, раннее из вершины под номером 4 в вершину под номером 3 уходило 4 единицы потока, но они были перенаправлены из вершины номер 4 к вершине номер 6. Однако из-за этого часть потока убывает из вершины под номером 5 (т. к. ранее она приходила из вершины под номером 3). Для восстановления баланса увеличивается напор на 1-3 на 4 единицы.

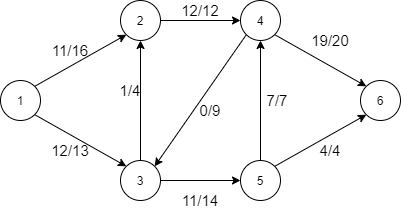


Рисунок 2.10

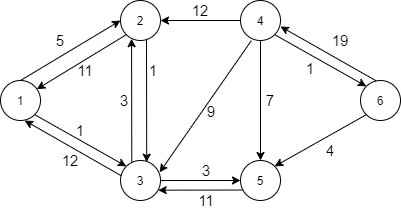


Рисунок 2.11

Введем вспомогательные **понятия разреза сети, потока через разрез, а также пропускной способности разреза.**

**Разрез сети G = (V, E)** с **истоком s** и **стоком t**, и **потоком f** – это разбиение множества **V** на два таких подмножества **S** и **T**, что **S = V – T**, причем **s  S** и **t  T**.

**Поток f(S, T)** через разрез сети определяют следующим образом:

**Пропускная способность разреза c(S, T)** =

Найдем три вышеперечисленные характеристики на примере сети, представленной на рисунке 2.12.

1. Разрез сети (S-T разрез):

S = { 1, 2, 3 },

T = { 4, 5, 6 }.

1. Поток f(S, T):

f(S, T) = f(2, 4) + f(3, 5) – f(4, 3) = 12 + 11 – 4 = 19.

1. Пропускная способность разреза c(S, T):

c(S, T) = c(2, 4) + c(3, 5) = 12 + 14 = 26;

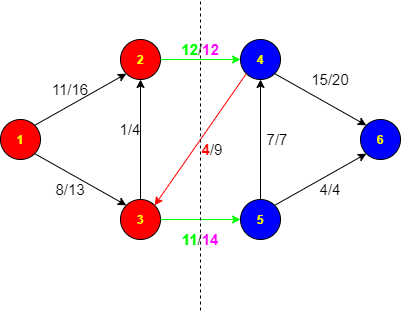


Рисунок 2.12

**Минимальный разрез сети** – это разрез, пропускная способность которого является наименьшей среди всех остальных возможных разрезов сети.

Используя все введенные понятия, сформулируем **теорему Форда-Фалкерсона:**

*Для некоторой сети G(V, E) с потоком f будут эквивалентны следующие утверждения:*

*1. f – максимальный поток в G.*

*2. Остаточная сеть в не содержит дополняющих путей.*

*3. Величина потока |f| равна величине пропускной способности минимального разреза сети G(V, E).*

**Теорема Форда-Фалкерсона** лежит в основе многих алгоритмов для поиска максимального потока в сети. В основном она служит «сигналом», что работу алгоритма можно прекратить. Если в остаточной сети не содержится дополняющих путей – работа алгоритма может быть приостановлена.

Благодаря **теореме Форда-Фалкерсона** можно найти максимальный поток в сети, перебирая все возможные разрезы (S, T) сети. Максимальный поток будет равен величине пропускной способности минимального разреза. Так для сети на рисунке 2.13 максимальный поток будет равен 12. Однако это далеко не самый эффективный алгоритм.

С учетом всех собранных сведений, терминов и понятий можно составить псевдокод **алгоритма Форда-Фалкерсона**.

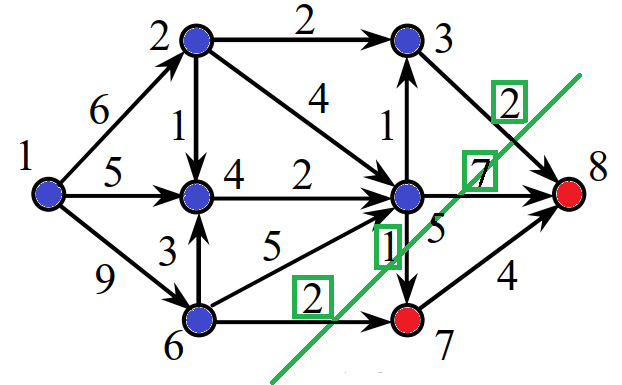
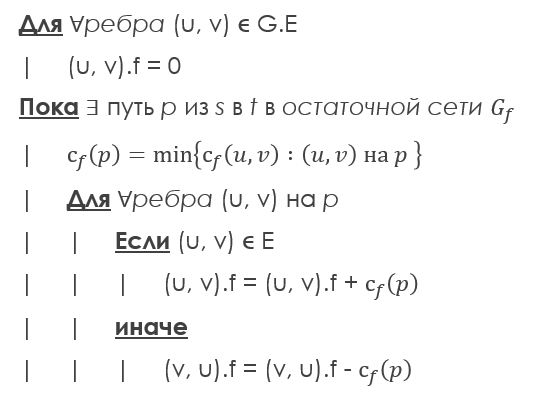


Рисунок 2.13

Псевдокод **алгоритма Форда-Фалкерсона**:



**2.2 Схема алгоритма Форда-Фалкерсона**

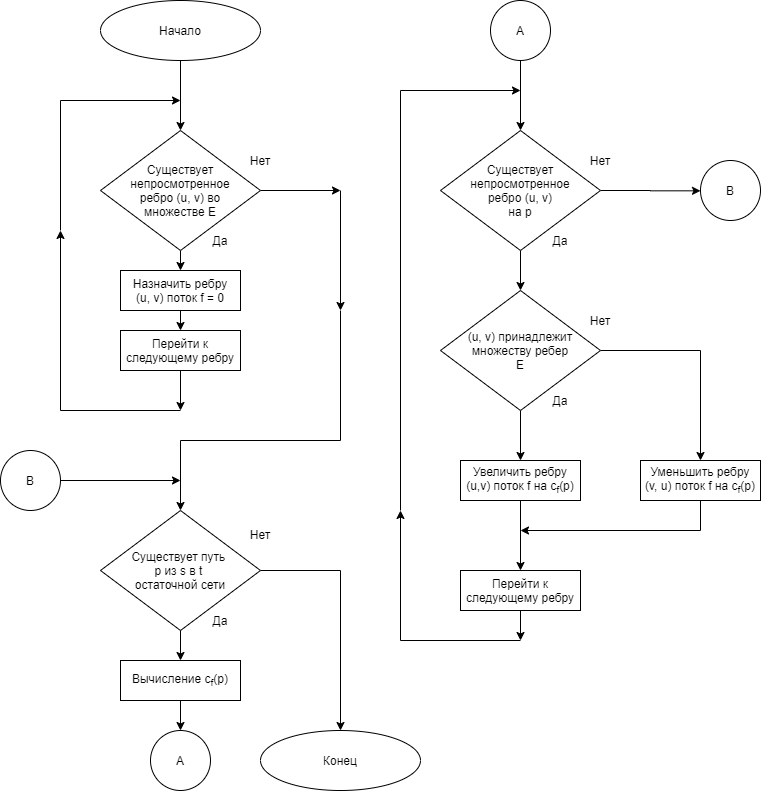
****

Рисунок 2.14

**2.3 Оценка сложности и эффективности алгоритма**

Рассмотрим сеть, представленную на рисунке 2.15.

В алгоритме Форда-Фалкерсона дополняющий путь в остаточной сети находится совершенно произвольным способом (к примеру, поиском в глубину). Вполне возможно, что алгоритм выберет дополняющий путь сначала как 1-2-3-4, а затем 1-3-2-4. Может быть, что такое происходить будет каждый раз. В итоге будет выполнено более 900 итераций, что крайне нежелательно.

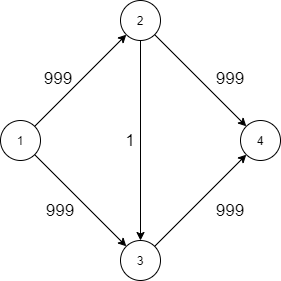


Рисунок 2.15

Таким образом можно сделать вывод, что сложность алгоритма Форда-Фалкерсона составляет O(f\*E), где E – число ребер в графе, а f – максимальный поток.

**Алгоритм Эдмондса-Карпа** абсолютно идентичен **алгоритму Форда-Фалкерсона**, только путь ищется не произвольный, а самый короткий. Ведь, чем короче путь, тем меньше вероятность встретить очень маленькое число, которое бы замедляло работу алгоритма.

Поскольку для нахождения наикратчайшего пути мы будем использовать алгоритм Дейкстры, то сложность алгоритма Эдмондса-Карпа равна O(V\*E2), где V – кол-во вершин, а Е – кол-во ребер.

**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Логическое разделение структуры программы**

Пользуясь принципами процедурного программирования разделим программу на два логических блока: на блок, отвечающий за непосредственно алгоритмическую часть программы (алгоритм Эдмондса-Карпа), а также на блок, ответственный за визуальную составляющую программы (графический интерфейс пользователя).

Рассмотрим в следующем подразделе в качестве первого блок, ответственный за нахождение максимального потока введенной сети.

**3.2 Файлы EdmondKarpAlgorithm.h и EdmondKarpAlgorithm.cpp**

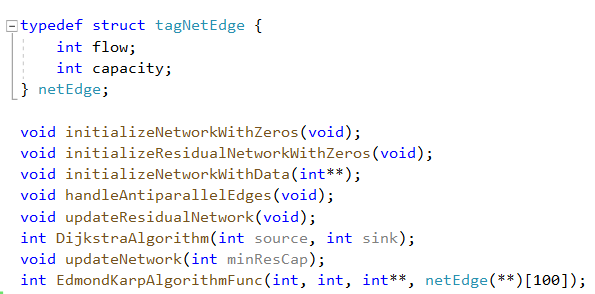


Рисунок 3.1 – Содержимое файла EdmondKarpAlgorithm.h

Файл EdmondKarpAlgorithm.h содержит в себе прототипы всех используемых функций для нахождения максимального потока сети. Рассмотрим их назначение.

Функция initializeNetworkWithZeros(void) обнуляет поток и пропускную способность по всей сети.

Функция initializeResidualNetworkWithZeros(void) обнуляет остаточную сеть по всем ребрам.

Функция initializeNetworkWithData(int\*\*) получает в качестве параметра указатель на указатель на тип int, в данном контексте это будет многомерный массив с M строками и тремя столбцами. Нулевой столбец будет содержать вершину, из которой исходит ребро сети, первый столбец будет содержать вершину, в которую будет входить ребро сети, а во втором столбце будет находиться пропускная способность ребра. Данная функция инициализирует матрицу смежности необходимыми данными.

Функция handleAntiparallelEdges(void) обрабатывает исключительную ситуацию с противоположно направленными ребрами, которых быть не должно согласно нашему определению сети.

Функция updateResidualNetwork(void) обновляет остаточную сеть с учетом текущего состояния основной сети. Вызывается во время работы алгоритма.

Функция DijkstraAlgrorithm(int source, int sink) находит наикратчайший путь из вершины с номером source в вершину с номером sink при помощи алгоритма Дейкстры. В качестве возвращаемого параметра выступает минимальная пропускная способность на найденном наикратчайшем пути.

Функция updateNetwork(int minResCap) обновляет текущую сеть. Параметр minResCap – это минимальная пропускная способность на найденном наикратчайшем пути в остаточной сети. Вызывается во время работы алгоритма.

Функция EdmondKarpAlgorithmFunc(int, int, int\*\*, newEdge(\*\*)[100]) является стартовой точкой для начала работы алгоритма Эдмондса-Карпа. Именно эта функция вызывается в другом файле, Source.cpp, отвечающем за GUI программного средства. Функция в качестве первого int параметра принимает число вершин в сети, второго int параметра принимает число дуг в сети, третьего int\*\* параметра принимает массив типа «вершин-источник»-«вершина-приемник»-«пропускная способность», четвертого newEdge(\*\*)[100] параметра принимает адрес указателя на массив из 100 экземпляров структуры newEdge. Этот параметр назначен именно таким образом, чтобы из функции можно было «вернуть» два результата одновременно – один из них возвращается традиционным способом, посредством оператора return (так возвращается максимальный поток сети), другой результат – это многомерный массив newEdge, хранящий распределенные по дугам единицы потока. В коде Source.cpp хранится указатель, в котором, по завершении работы EdmondKarpAlgorithmFunc(…), будет находиться адрес многомерного массива newEdge. Таким образом, в коде Source.cpp у нас будет доступ к массиву, объявленному в EdmonKarpAlgorithm.cpp и хранящему результат работы алгоритма.

**3.3 Тестирование EdmondsKarpAlgorithm.cpp**

Перед тем, как перейти к созданию визуальной составляющей программного средства, необходимо было убедиться в корректности работы алгоритмической части. Для этого код исходной программы был выслан на тестирование на сайт informatics.mccme.ru.

Единственным отличием высылаемого решения от финальной версии кода, используемой в программном средстве, это наличие функции main вместо функции EdmondKarpAlgorithmFunc, так как тестирующая система предполагает, что программа принимает данные из потоков stdin и stdout через консоль.

Программа успешно прошла тестирование, как видно на рисунке 3.3.

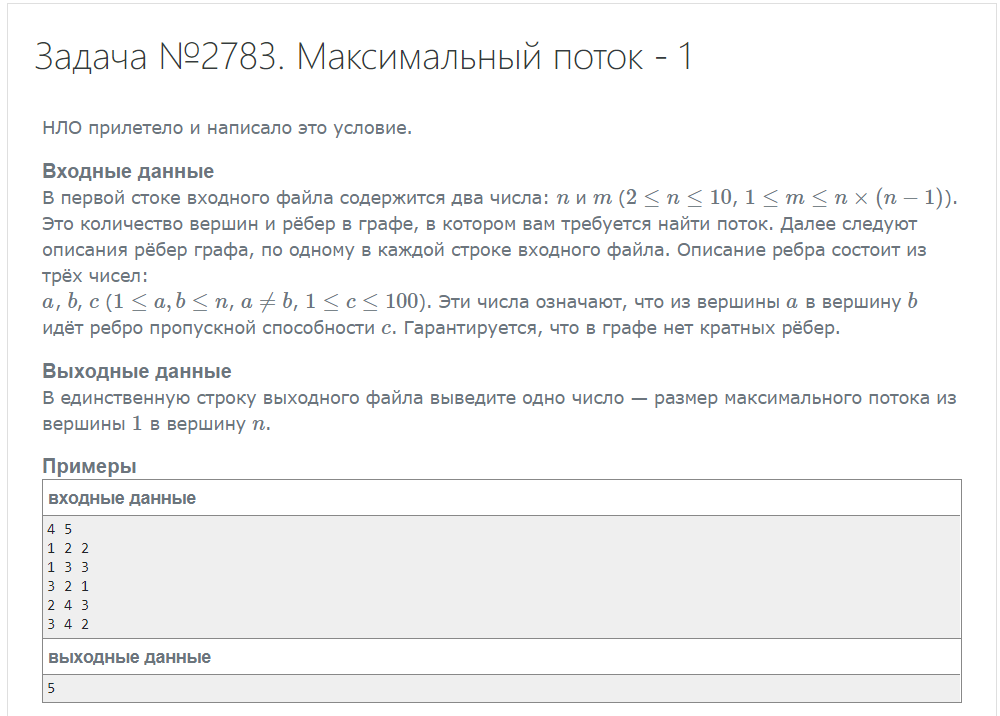


Рисунок 3.2 – Скриншот с условием задачи с сайта informatics.mccme.ru

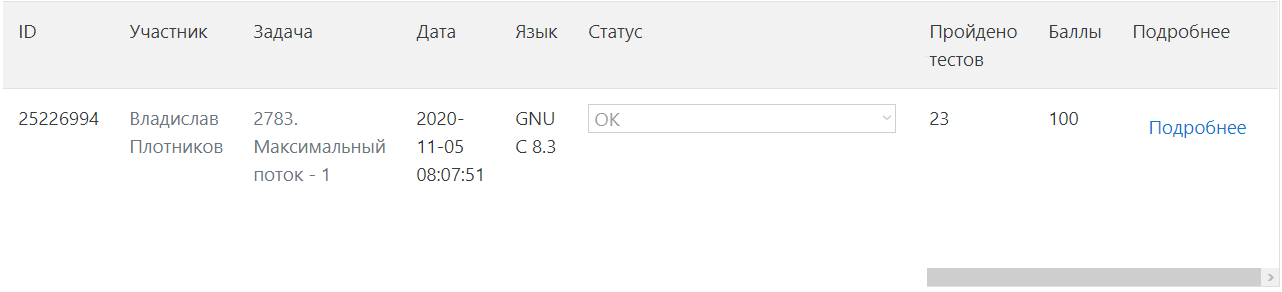


Рисунок 3.3 – Скриншот с принятием решения тестирующей системой

**3.4 Файл Source.cpp**

Данный файл содержит большое количество функций, многие из которых выполняют довольно рутинную работу и не требуют детального пояснения.

Однако, поскольку GUI было создано при помощи WinAPI, две функции в данной программе вызывают достаточно большой интерес – это функции WinMain и MyWindowProc.

WinAPI (Windows Application Programming Interface) – это набор функций и инструментов, позволяющих программисту создавать приложения (программы), работающие в операционной системе Windows.

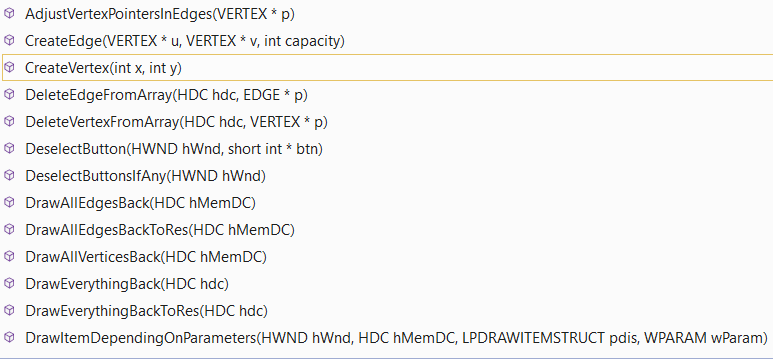


Рисунок 3.4 – Прототипы функций файла Source.cpp – Часть 1

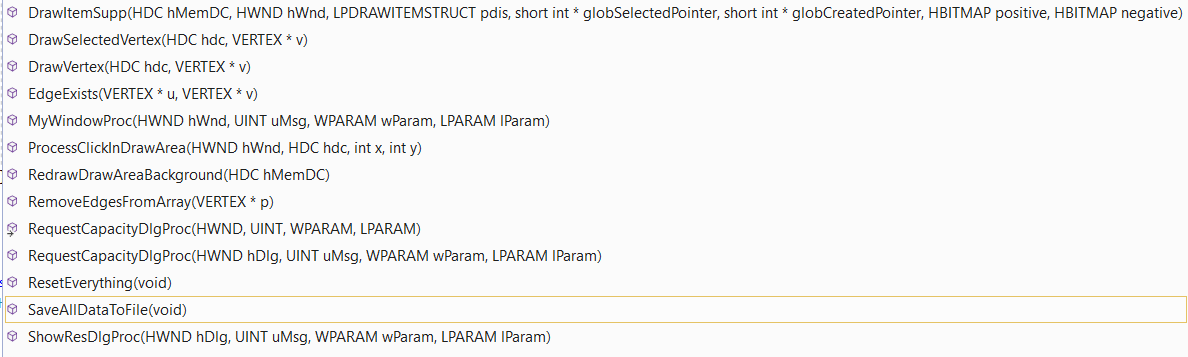


Рисунок 3.5 – Прототипы функций файла Source.cpp – Часть 2

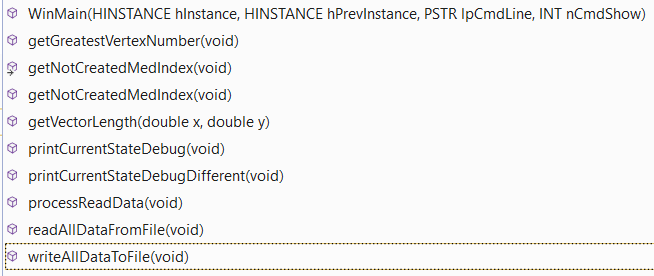


Рисунок 3.6 – Прототипы функций файла Source.cpp – Часть 3

Основой взаимодействия программы с пользователем и с операционной системой является концепция сообщений. Сообщение – это структура данных, которая содержит такие элементы как:

* дескриптор окна, которому адресовано сообщение;
* код (номер) сообщения;
* дополнительную информацию, которая зависит от кода сообщения.

Поскольку Windows – многозадачная операционная система, то все поступающие внешние сообщения приходят только одному из работающих приложений, а точнее – активному окну. Windows, в свою очередь, является диспетчером сообщений. С момента запуска ОС создает глобальный объект в памяти, называемый системной очередью сообщений. Абсолютно все сообщения, генерируемые, как и аппаратурой, так и приложениями, помещаются в эту очередь. Операционной системой эта очередь периодически опрашивается, в случае, если она не пуста, Windows посылает сообщение адресату, который определяется дескриптором окна.

Сообщения, которые получает приложение, могут поступать асинхронно из различных источников. К примеру, приложение может зависеть от работы системного таймера, которое, в свою очередь, посылает сообщения с определенным интервалом приложению, и в то же время приложение должно быть готовым в любой момент получить любое сообщение от операционной системы. Дабы предотвратить утечку сообщений, Windows в момент запуска приложения создает еще один глобальный объект, называемый очередью сообщений приложения. Этот объект живет до тех пор, покуда живет само приложение.

Следовательно, путь следования сообщений можно представить схемой на рисунке 3.7.



Рисунок 3.7

Теперь необходимо дать определение понятию «оконная процедура».

Оконная процедура – это функция обратного вызова, предназначенная для обработки сообщений, адресованных любому окну того оконного класса, в котором содержится ссылка на данную процедуру.

Функции обратного вызова – это функции, вызываемые самой операционной системой, помечаются спецификатором CALLBACK. Оконная процедура имеет следующим прототип:



Используемая нами оконная процедура MyWindowProc в программе имеет такой же прототип. Параметр hWnd – это дескриптор окна, которому адресуется сообщение. Параметр uMsg – это код сообщения, wParam и lParam - это параметры сообщения.

В теле оконной процедуры обычно пишется оператор switch, внутри которого и происходит обработка нужных сообщений. У каждого сообщения в операционной системе Windows есть соответствующий символьный идентификатор. Все идентификаторы системы заданы с помощью директивы #define в заголовочном файле winuser.h.

Наиболее часто подвергаются обработке оконные сообщения, имеющие префикс WM\_, в качестве примера: WM\_MOVE, WM\_PAINT, WM\_SIZE и т.д. Теперь необходимо дать определение термину «оконный класс».

Оконный класс (он же класс окна) – это структура, которая определяет главные характеристика окна. К таким характеристикам относят стиль окна и ресурсы, например, пиктограммы, кисти и меню. Помимо этого, одним из полей структуры является адрес оконной процедуры, которая будет ответственна за обработку сообщений, получаемых любым окном данного оконного класса. Именно ссылка на оконный класс передается функции CreateWindow, вызываемой с целью создать окно. Использование оконного класса позволяет создавать множество окон на основе одного и того же класса и, соответственно, с одной и той же оконной процедурой. Ярким примером подобного использования является оконный класс BUTTON. Оконная процедура данного класса расположена в динамически подключаемой библиотеке, и она ответственна за обработку сообщений для всех кнопок всех окон. Похожие подобные классы существуют и для других элементов управления, например, для списков и полей редактирования. В своей совокупности, все эти классы называются стандартными или предопределенными оконными классами. Также Windows содержит готовый оконный класс и для диалоговых окон, играющих достаточно важную роль в графическом интерфейсе пользователя. Для главного окна нашего приложения был создан собственный класс окна.

Однако, ни в коему случае не следует путать понятия оконного класса Windows и класса в объектно-ориентированных языках программирования, поскольку исторически концепция оконных классов появилась раньше концепции класса в области ООП.

Необходимым компонентом всех приложений для Windows является цикл обработки сообщений. У всех приложений всегда есть главная функция WinMain. Именно в ней содержатся вызовы функций инициализации и создания окон, за которыми идет цикл обработки сообщений и необходимый код для завершения работы приложения. Наше программное средство не было исключением.

Windows записывает все сообщения, которые были адресованы приложению, в очередь сообщений приложения. Для извлечения сообщения из этой очереди используется функция GetMessage. В случае, если сообщение имеет код WM\_QUIT, то осуществляется выход из цикла, на этом работа приложения завершается.

Если извлеченное сообщение не было сообщением WM\_QUIT, то оно передается для обработки функции DispatchMessage, таким образом сообщение посылается обратно в Windows. После этого Windows перенаправляет сообщение для обработки назначенной оконной процедуре, т.е. происходит вызов оконной процедуры операционной системой Windows. Как только происходит возврат из оконной процедуры, управление передается оператору, расположенному после DispatchMessage, работа цикла продолжается.

Обобщая всё вышесказанное, программа на WinAPI содержит как минимум две функции:

* WinMain – главная функция, точка входа в приложение. Здесь создается главное окно программы и происходит запуск цикла обработки сообщений;
* WndProc – оконная процедура, главная задача которой – обеспечить обработку сообщений для основного окна программы.

Схема работы двух функций представлена на рисунке 3.8.

Сразу после входа в функцию WinMain создается и регистрируется класс главного окна приложения. Для этого создается экземпляр структуры WNDCLASSEX wcex и заполняются её поля. Всего эта структура имеет 12 полей, как представлено на рисунке 3.9.

После этого происходит вызов функции RegisterClassEx(&wcex), которая принимает адрес созданного нами оконного класса. После вызова оконный класс становится зарегистрированным.

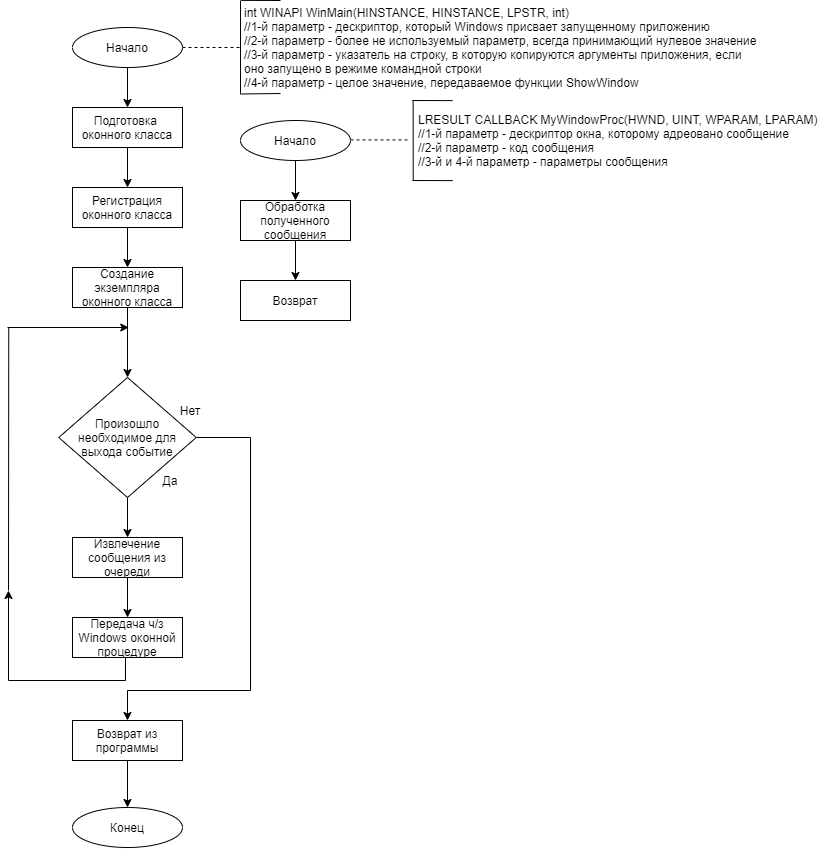


Рисунок 3.8 – Схема работы приложения WinAPI

По успешной регистрации класса окна, следующий этап – это создание главного окна приложения. Для этого вызывается функция CreateWindowEx, прототип которой очень схож с прототипом другой функции, CreateWindow, представленным на рисунке 3.10.

Все параметры функции CreateWindowEx имеют тот же смысл, что и у функции CreateWindow. Единственное отличие – это наличие первого параметра dwExStyle, задающего расширенный стиль окна, который применяется совместно со стилем, определяемым параметром dwStyle (из расширенных стилей – утопленный край, возможность принимать перетаскиваемые файлы, наличие трехмерной рамки, прозрачность окна и др.)

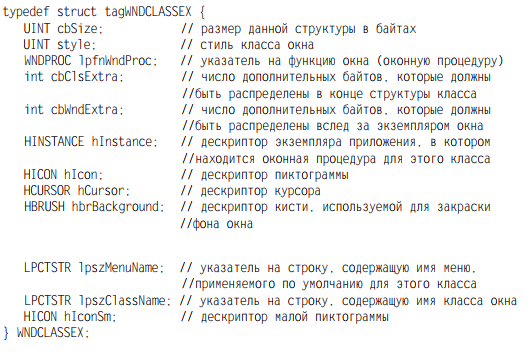


Рисунок 3.9 – Поля структуры WNDCLASSEX

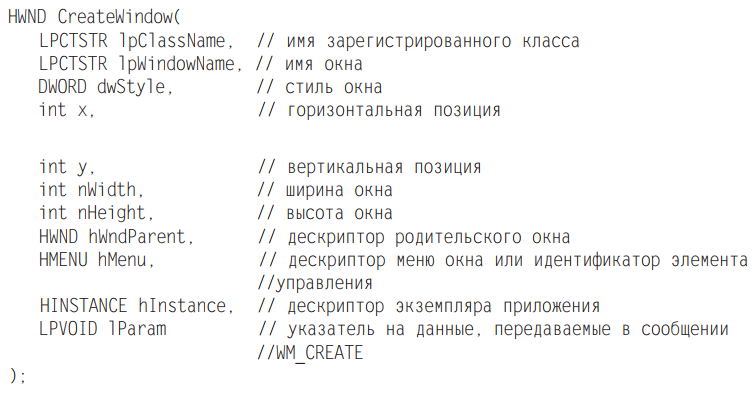


Рисунок 3.10 – Прототип функции CreateWindow

Помимо основного окна приложения через функцию CreateWindowEx создаются все элементы управления окна – кнопка SRC, кнопка SNK, кнопка для промежуточных вершин, кнопка расчета максимального потока, а также все надписи. Отличительной особенностью вызова CreateWindowEx в этих случаях является передача дескриптора основного окна приложения (возвращенного функцией CreateWindowEx при самом первом вызове) в качестве параметра. Таким образом Windows распознает, что все создаваемые элементы управления являются дочерними по отношению к главному окну приложения.

Однако, созданное окно необходимо отобразить на экране, именно поэтому происходит вызов функции ShowWindow, которой в качестве первого параметра передается дескриптор главного окна приложения, а в качестве второго параметра – последний параметр функции WinMain (он ответственен за то, каким образом будет показано окно).

После того, как окно отображено на экране, необходимо начать цикл обработки сообщений (один из возможных вариантов цикла обработки сообщений представлен на рисунке 3.11).

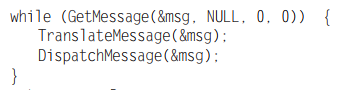


Рисунок 3.11 – Цикл обработки сообщений

Здесь, msg – это экземпляр структуры типа MSG, используемая для получения и отправки сообщений в очередь сообщений приложения. Поля данной структуры представлены на рисунке 3.12.

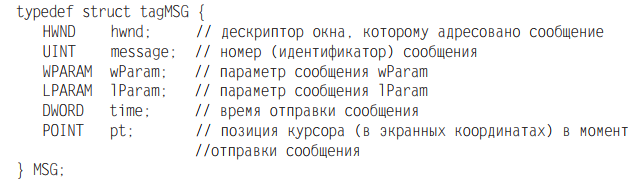


Рисунок 3.12 – Поля структуры MSG

Извлечение очередного сообщения происходит благодаря вызову функции GetMessage, обладающей следующим прототипом:



Рисунок 3.13 – Прототип функции GetMessage

Первый параметр задает адрес структуры типа MSG, в которую помещается выбранное сообщение.

Второй параметр hWnd – это дескриптор окна, принимающего сообщение. В случае, если этот параметр равен NULL, то происходит выборка сообщений для всех окон приложения.

С помощью третьего параметра wMsgFilterMin можно указать минимальный номер принимаемого сообщения, обычно ему присваивают нулевое значение.

Аналогичное назначение у четвертого параметра wMsgFilterMax, только он указывает максимальный номер принимаемого сообщения. В случае, если оба параметра равны нулю, то обработке подвергаются абсолютно все сообщения из очереди.

Функция GetMessage будет возвращать значение TRUE до тех пор, покуда не будет извлечено сообщение WM\_QUIT. При получении сообщения WM\_QUIT, функция вернет значение FALSE. Как итог, происходит выход из цикла, и приложение завершает работу, а операционной системе возвращается код возврата из msg.wParam.

В случае, если извлечено было не сообщение WM\_QUIT, то происходит выполнение тела цикла. В теле цикла указан вызов двух функций – TranslateMessage и DispatchMessage, причем обе в качестве параметра получают адрес экземпляра структуры MSG.

На самом деле, вызов TranslateMessage необходим в тех приложениях, в обязанность которых входит обработка ввода данных с клавиатуры. TranslateMessage вызывается с целью обеспечить независимость приложения от аппаратных платформ и различных национальных клавиатурных раскладок. Для этого в Windows существует реализация схема обработки сообщений от символьных клавиш, состоящей из двух уровней. То есть, система сначала генерирует сообщения о виртуальных клавишах, а после вызова TranslateMessage в очередь помещается сообщение WM\_CHAR, которое представляет собой преобразованное сообщение о виртуальной клавише. На следующей итерации цикла (на втором этапе) происходят извлечение сообщения WM\_CHAR и последующая его обработка. Однако наше приложение не будет принимать зависящий от платформы или раскладки ввод, по этой причине вызов TranslateMessage в нашей реализации опущен. Согласно документации MSDN [9], такая реализация цикла обработки сообщений допустима.

В свою очередь, вызовом функции DispatchMessage происходит передача структуры msg обратно в Windows. Затем, операционная система для обработки этого сообщения вызывает функцию обратного вызова, она же соответствующая оконная процедура.

Важно отметить, что сообщение передается именно той оконной процедуре, которая была указана в поле структуры оконного класса, экземпляром которого является окно, которому было адресовано сообщение.

Уже после возврата из оконной процедуры управление будет передано оператору, стоящему после DispatchMessage, то есть ход программы вернется на начало цикла while. Таким образом, работа продолжается.

В оконной процедуре уже происходит реальная работа программного средства. Именно оконной процедурой определяется, что выводится в клиентскую область окна, а также то, как программа реагирует на пользовательский ввод.

В оконной процедуре обычно располагают крупный switch, в котором проверяется значение переменной типа UINT (второй по счету параметр оконной процедуры). Документацией MSDN [9] совершенно точно определяется, каким образом нужно обрабатывать сообщение определенного типа и какое возвращение должна возвращать оконная процедура. Абсолютно все сообщения в оконной процедуре нашего программного средства были обработана, как того требует документация, поскольку ненадлежащая обработка сообщений может привести к неопределенному поведению или потере ресурсов системы. Подобные ситуации недопустимы.

Если же нет необходимости в обработке получаемой оконной процедурой сообщения, то сообщение должно быть передано функции DefWindowProc, а значение, возвращаемое этой функцией, должно быть возвращено оконной процедурой. DefWindowProc предоставляет «обработку по умолчанию» всех сообщений, не обрабатываемых программистом собственноручно. То есть программист пишет код только для сообщений, нуждающихся в нестандартной обработке, например, WM\_CREATE, WM\_CLOSE, WM\_DESTROY, WM\_PAINT, WM\_KEYDOWN и т.д.

Очень важной является обработка сообщения WM\_PAINT в оконной процедуре, так как именно это сообщения даёт уведомление программе, что часть или даже вся клиентская область окна стала недействительной и её срочно необходимо перерисовать. Вся клиентская область окна является недействительной при создании окна, так как в ней еще ничего на нарисовано. В случае, если меняются размеры окна, клиентская область также является недействительной. Недействительной клиентская область окна становится также при полном сворачивании и разворачивании программы. Недействительность клиентской области окна проявляет себя так же в случае, если окно приложения было перекрыто окном другого приложения, либо же при перемещении окна тогда, когда часть окна вышла за пределы экрана компьютера.

Обработка сообщения WM\_PAINT всегда должна начинаться с вызова функции BeginPaint(HWND, LPPAINTSTRUCT). Первый параметр – это дескриптор окна, полученный посредством аргумента оконной процедуры, а второй – это адрес структуры типа PAINTSTRUCT. В результате вызова функции BeginPaint поля структуры PAINSTRUCT становятся заполненными данными, которые в дальнейшем будут использованы операционной системой. Особо важный интерес для нас представляет значение, возвращаемое функцией BeginPaint – это дескриптор контекста устройства.

Контекст устройства описывает некоторое физическое устройство вывода информации, такие как дисплей или принтер, представляет собой важнейший объект графической подсистемы Windows. Другими словами, контекст устройства – это некоторая внутренняя структура данных, хранящая наиболее часто используемые графические атрибуты, например, перо, шрифт, кисть, цвет фона и т.д.

Необходимо отметить два эффекта, достигаемых при вызове функции BeginPaint. Первый эффект – во время обработки вызова BeginPaint, операционная система Windows обновляет фон клиентской области, если регион, требующий обновления, помечен для стирания. В качестве кисти по умолчанию используется кисть, обозначенная в поле hbrBackground структуры WNDCLASSEX. Второй эффект – вся клиентская область становится действительной, данный эффект предотвращает повторное генерирование системой сообщения WM\_PAINT бесконечное множество раз.

Также в WM\_PAINT может следовать вызов функции GetClientRect(HWND, LPRECT) (в нашем коде она не использовалась, но достаточно много можно встретить использование структуры, указатель на которую передается вторым параметром этой функции). Первым параметром является дескриптор окна, результат вызова функции помещается в структуру, адрес которой передавался вторым параметром – это структура типа RECT. Ознакомимся с полями структуры RECT на рисунке 3.14.

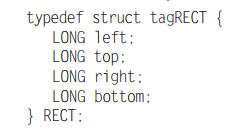


Рисунок 3.14 – поля структуры RECT

Поля этой структуры задают координаты левого верхнего угла и правого нижнего угла прямоугольника. После выполнения функции GetClientRect, left и top всегда получают нулевое значение, а right и bottom будут содержать ширину и высоту клиентской области окна соответственно. Данная структура использовалась при рисовании практически каждого элемента управления приложения: кнопки, области рисования, рамки и вершины сети.

Вся разработка приложения сводилась к аккуратной обработке каждого необходимого сообщения, с выделением памяти под необходимые ресурсы, а также обязательным освобождением памяти, дабы избежать утечки памяти либо же переполнения стека, поскольку подобные явления могут оказать самое негативное влияние на производительность программы – может возникнуть ситуация, что программе больше невозможно будет выделить ресурсы, либо же перерисовка окна приложения начнет давать сбои – появятся блики и мерцания и др.

Отдельного упоминания заслуживает буферизация вывода на экран. Дело в том, что все элементы, расположенные на экране приложения, можно было бы рисовать последовательно – то есть сначала рисовать фон, затем рамку, затем одну за другой вершины, дуги сети и другие элементы интерфейса. Однако, с таким подходом к рисованию, неизбежна проблема мерцания окна, поскольку Windows GDI (Graphics Device Interface) приходится каждый раз брать инструмент, рисовать некоторую часть окна, класть инструмент обратно, освобождать под него, выделять снова и т.д. С целью избежать этого, используется буферизация.

Суть буферизации заключается в том, что сначала в памяти компьютера выделяется место для некоторой прямоугольной области заданного размера. Затем уже непосредственно в памяти Windows GDI формирует полную целую картину того, что необходимо будет вывести. При этом системе нет необходимости переключаться с непосредственного рисования на вывод и обратно. После того как в памяти создана картинка типа BITMAP, она, будучи готовой, единожды выводится на экран. Таким образом, решается проблема мерцания экрана, и приложение может рисовать до 20 вершин и 100 дуг на экране без потери производительности – перемещение вершин достаточно плавное, а окно не мерцает (рисунок 3.15). Реализация буферизации представлена в листинге 3.1.

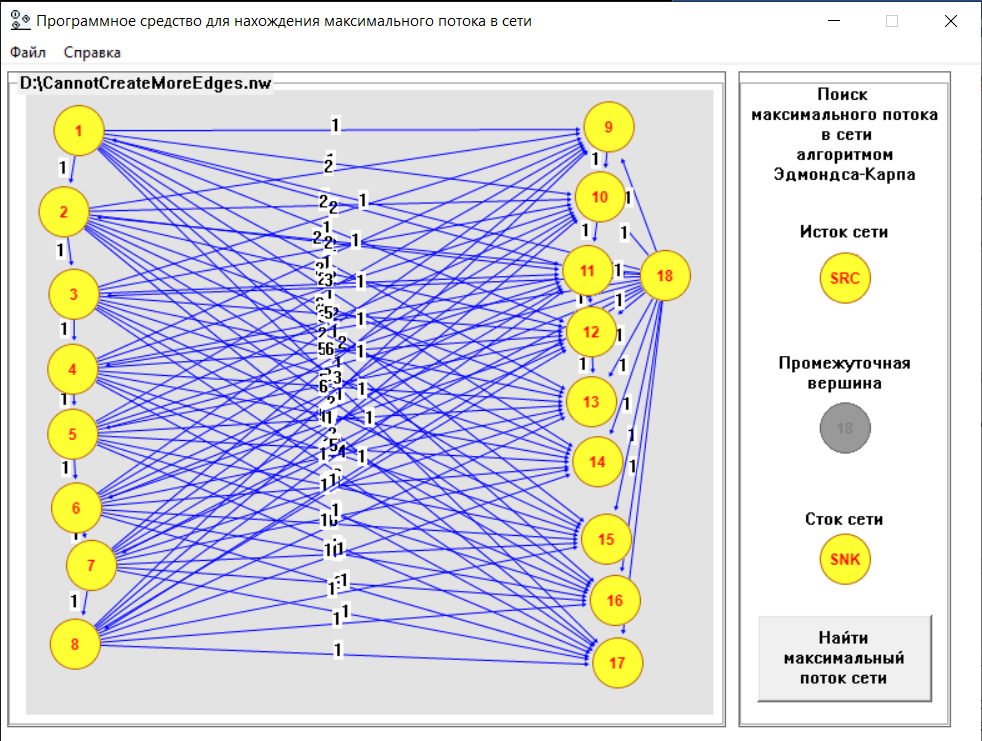


Рисунок 3.15 – Окно приложения с сотней дуг

void DrawEverythingBack(HDC hdc) {

HDC hMemDC = CreateCompatibleDC(hdc);

HBITMAP memBM = CreateCompatibleBitmap(hdc, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT);

SelectObject(hMemDC, memBM);

RedrawDrawAreaBackground(hMemDC);

DrawAllEdgesBack(hMemDC);

DrawAllVerticesBack(hMemDC);

BitBlt(hdc, DRAW\_AREA\_UPPER\_LEFT\_X, DRAW\_AREA\_UPPER\_LEFT\_Y, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT, hMemDC, 0, 0, SRCCOPY);

DeleteDC(hMemDC);

DeleteObject(memBM);

}

Листинг 3.1 – Основная функция перерисовки

**3.5 Файл resource.h**

В программе используются заранее подготовленные ресурсы – так, вручную нарисованы кнопки SNK, SRC, а также все промежуточные вершины от 1 до 18 включительно, как активные, так и неактивные версии вышеперечисленного.

Так же среда разработки Visual Studio содержит встроенный редактор ресурсов для растровых изображений и для создания макетов диалоговых окон.

Диалоговые окна значительно облегчают разработку приложения, поскольку они не требуют создания отдельного оконного класса, а обработку большинства сообщений берет на себя операционная система Windows. Так, окно с выводом результата работы программы является диалоговым окном, а расположение всех элементов управления было задано редактором окон в Visual Studio (рисунок 3.16).

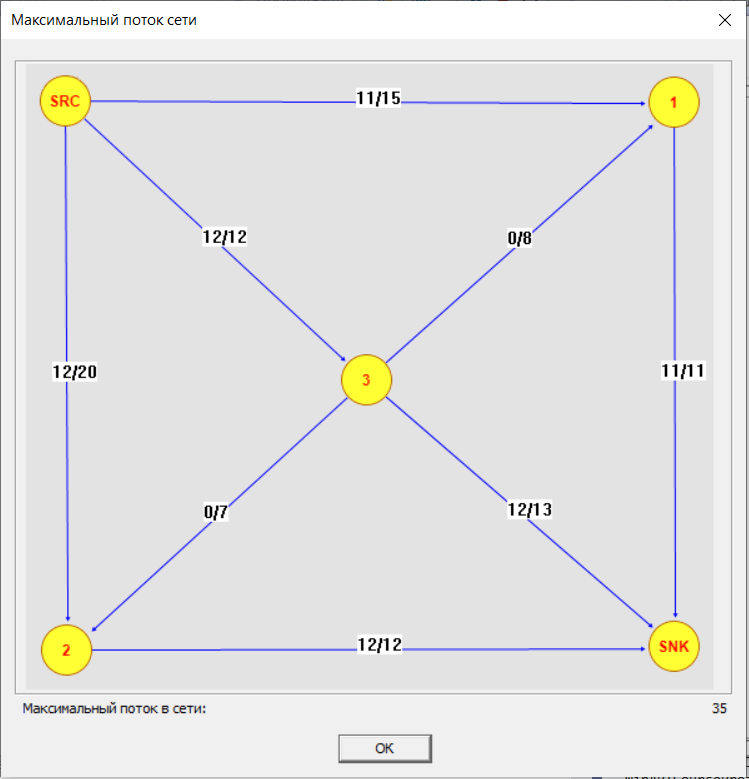


Рисунок 3.16 – Окно вывода результата программы

Так, файл resource.h автоматически создается Visual Studio, в нем хранятся уникальные идентификаторы каждого из ресурсов (объявляются директивой #define), тем самым позволяя их довольно удобно использовать в основном коде программы – Source.cpp (всем идентификаторам присвоены говорящие имена).

**4 ТЕХНИЧЕСКИЕ ПРИЁМЫ ПРОГРАММИРОВАНИЯ**

При написании программы строго выполнялись все рекомендации и требования, указанные в MSDN [9].

Поскольку, в отличие от таких языков, как Java, где есть уборщик мусора, в Си и написанных на нем программах ответственность за освобождение занятой памяти целиком и полностью лежит на программисте-разработчике (WinAPI в этом плане не исключение). По этой причине проводилось тщательное наблюдение за тем, чтобы все занимаемые ресурсы должным образом освобождались.

Где это было возможно, использовались гибкость и возможности языка Си, к примеру, при написании следующего фрагмента кода.

void DrawItemSupp(HDC hMemDC, HWND hWnd, LPDRAWITEMSTRUCT pdis,

short int\* globSelectedPointer, short int\* globCreatedPointer,

HBITMAP positive, HBITMAP negative) {

{

switch (pdis->itemAction) {

case ODA\_SELECT:

if ((pdis->itemState & ODS\_SELECTED) && !(\*globCreatedPointer)) {

(\*globSelectedPointer) = (\*globSelectedPointer) ? 0 : 1;

}

}

if ((\*globSelectedPointer)) {

SelectObject(hMemDC, negative);

}

else {

if (!(\*globCreatedPointer)) {

SelectObject(hMemDC, positive);

DeselectButton(hWnd, globCreatedPointer);

}

else

SelectObject(hMemDC, negative);

}

}

}

Листинг 4.1 – Функция DrawItemSupp

Прелесть этого участка кода заключается в том, самая суть функции – это перерисовка некоторого элемента управления, в зависимости от текущего состояния программы. Причем так выходит, что задача, решаемая данной функцией, одинакова для трех типов элементов (трех кнопок в нашем приложении). Однако, писать три функции для каждой кнопки, тела которых практически ничем не будут отличаться, - не самое рациональное решение.

Использование указателей (работа с адресами) позволило использовать один и тот же код для многих типов кнопок.

**5 ТЕСТИРОВАНИЕ**

Результаты тестирования программы представлены в таблице 5.1. Подтверждение успехов тестирования представлено на рисунках 5.1 – 5.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тест | Ожидаемый результат | Результат |
| 1 | **Запуск программы.**  1. Выделить файл WinAPICourseProject.exe.  2. Открыть его любым из возможных способов. | 1. Программа запускается.  2. Отображается поле для ввода сети, строка меню и кнопки для вершин сети.  3. Открыт файл «Безымянный.nw». | **Успех** |
| 2 | **Нахождение максимального потока некоторой заданной пользователем сети.**  1. Нажать кнопку выбора вершины-истока SRC.  2. Разместить вершину-исток SRC на сером поле ввода, нажав левой кнопкой мыши по нему.  3. Разместить все промежуточные вершины от 1 до 6 и конечную вершину SNK, руководствуясь теми же принципами.  4. Проставить дуги с пропускными способностями, выделяя кликом левой кнопки мыши вершину-источник, а затем вершину-приёмник и вводя корректные значения.  5. Нажать кнопку «Найти максимальный поток сети». | 1. Появление диалогового окна с выводом результата работы программы.  2. Ожидаемое значение максимального потока сети – 12. | **Успех** |
|  |  |  |  |
|  |  |  |  |
| 3 | **Сохранение введенной пользователем сети в файл.**  1. Нажать кнопку «Файл» в строке-меню приложения.  2. Нажать кнопку «Сохранить как…».  3. Выбрать имя и место для сохранения заданной сети.  4. Нажать кнопку «Сохранить». | 1. Появление диалогового окна с сообщением об успешной записи файла.  2. В выбранной директории появляется файл с расширением .nw с соответствующим именем. | **Успех** |
| 4 | **Закрытие программы.**  1. Нажать крестик в верхнем правом углу экрана либо же нажать кнопку «Выход» в меню «Файл». | 1. Программа успешно завершает свою работу. | **Успех** |
| 5 | **Открытие сохраненной пользователем сети.**  1**.** После запуска программы перейти в меню «Файл».  2. Нажать кнопку «Открыть».  3. Выбрать ранее сохраненный файл с сетью. | 1. Файл будет успешно прочитан.  2. Сеть без изменений будет отображена в сером поле ввода.  3. В строке с именем файла вместо «Безымянный.nw» будет указан полный путь к открытому файлу с расширением .nw. | **Успех** |
| 6 | **Удаление средней вершины из сети.**  1. Выделить левой кнопкой мыши некоторую промежуточную вершину.  2. Нажать клавишу «Backspace». | 1. Вершина исчезнет.  2. Исчезнут все входящие в вершину и исходящие из вершины дуги сети. | **Успех** |
| 7 | **Попытка найти максимальный поток для некорректно введенной сети.**  1. В сети должно выполняться одной из условий:   * не все пути ведут от истока в сток * отсутствие истока * отсутствие стока   2. Нажать кнопку «Найти максимальный поток сети». | 1. Диалоговое окно с выводом максимального потока не будет отображено.  2. Программа выведет предупреждение, сообщающее о проблеме в введенном графе. | **Успех** |
| 8 | **Попытка некорректного ввода пропускной способности дуги.**    1. Выполнить ранее описанные действия для ввода пропускной способности дуги.  2. Ввести некорректную пропускную способность (строковый литерал, отрицательное число, дробное число и др.). | 1. Программа не допустит ввода значения.  2. Вывод предупреждения. | **Успех** |
| 9 | **Попытка направить дуги в вершину-источник.**  1. Выделить промежуточную вершину.  2. Нажать на вершину-источник. | 1. Появление диалогового окна с вводом пропускной способности не произойдет.  2. Вывод предупреждения. | **Успех** |
|  |  |  |  |
|  |  |  |  |
| 10 | **Попытка направить дуги из вершины-стока.**  1**.** Выделить вершину-сток.  2. Нажать на промежуточную вершину. | 1. Появление диалогового окна с вводом пропускной способности не произойдет.  2. Вывод предупреждения. | **Успех** |

Таблица 5.1 – Case-тесты



Рисунок 5.1

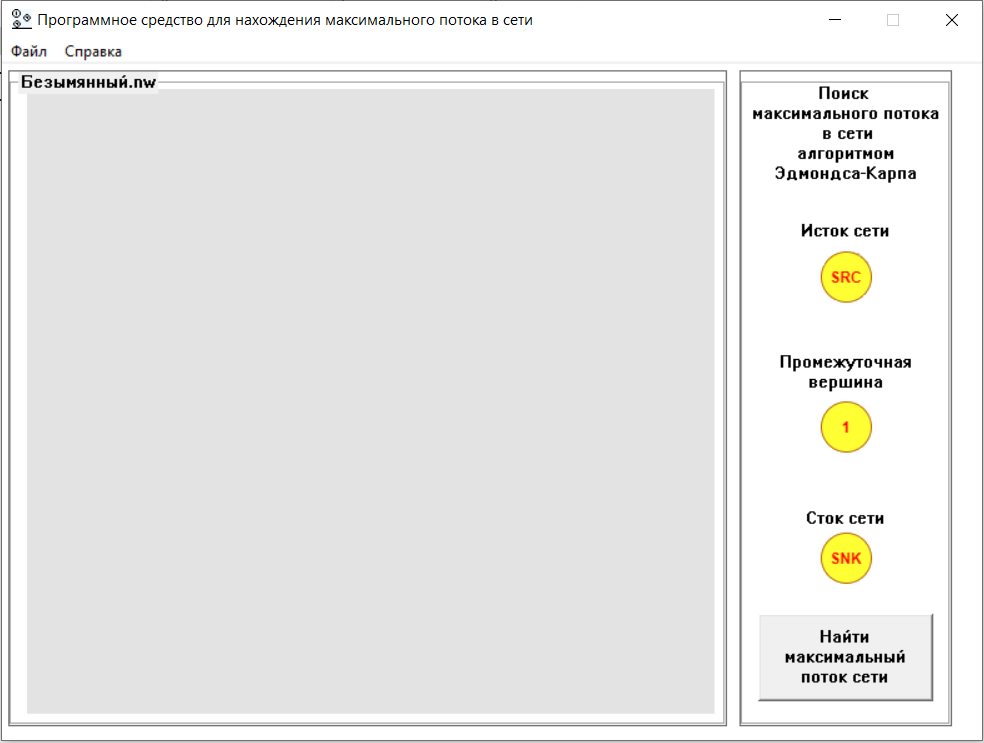


Рисунок 5.2

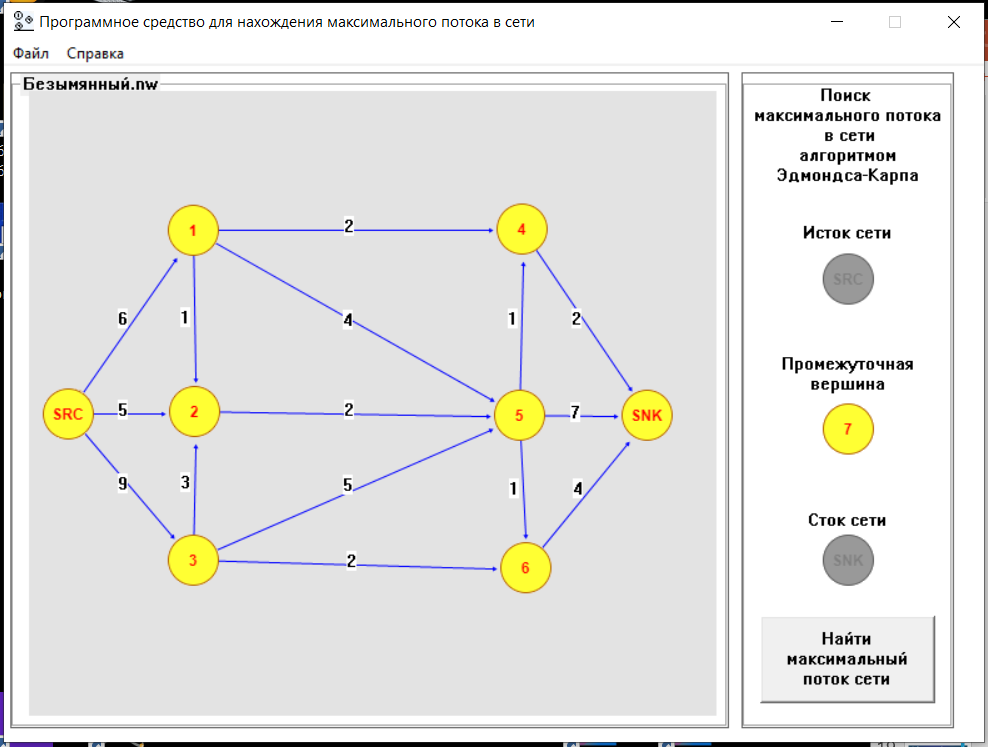


Рисунок 5.3

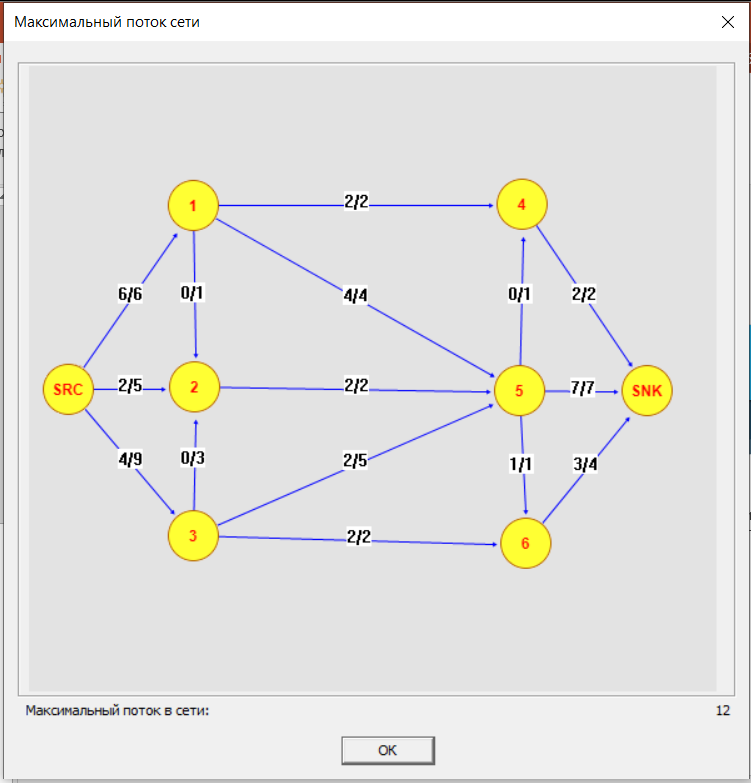


Рисунок 5.4

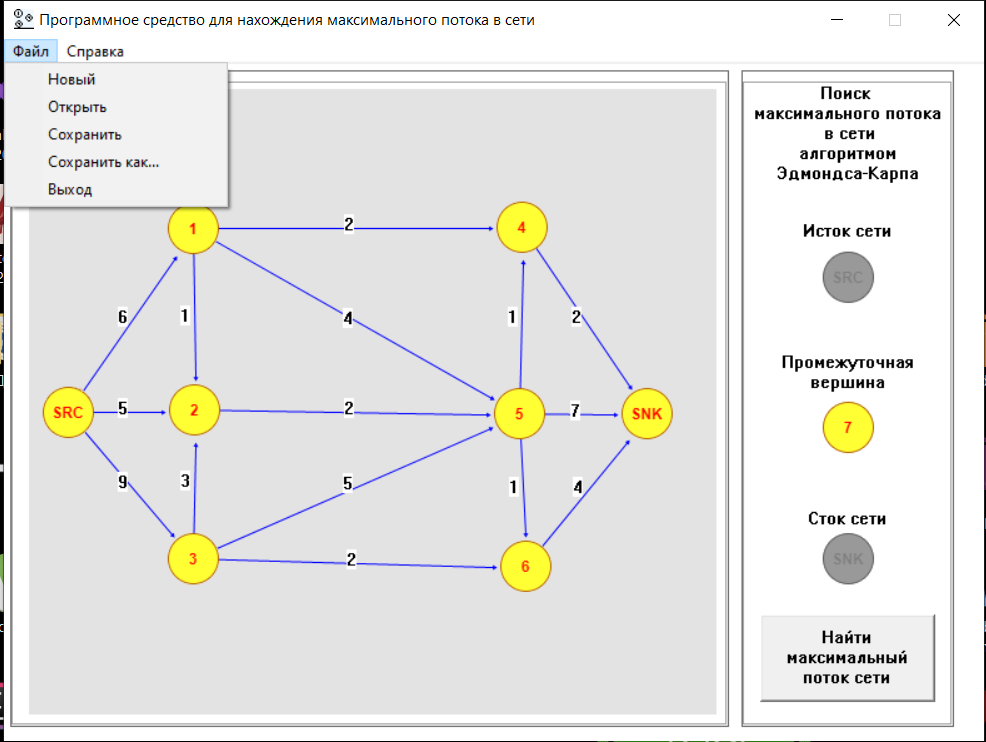


Рисунок 5.5

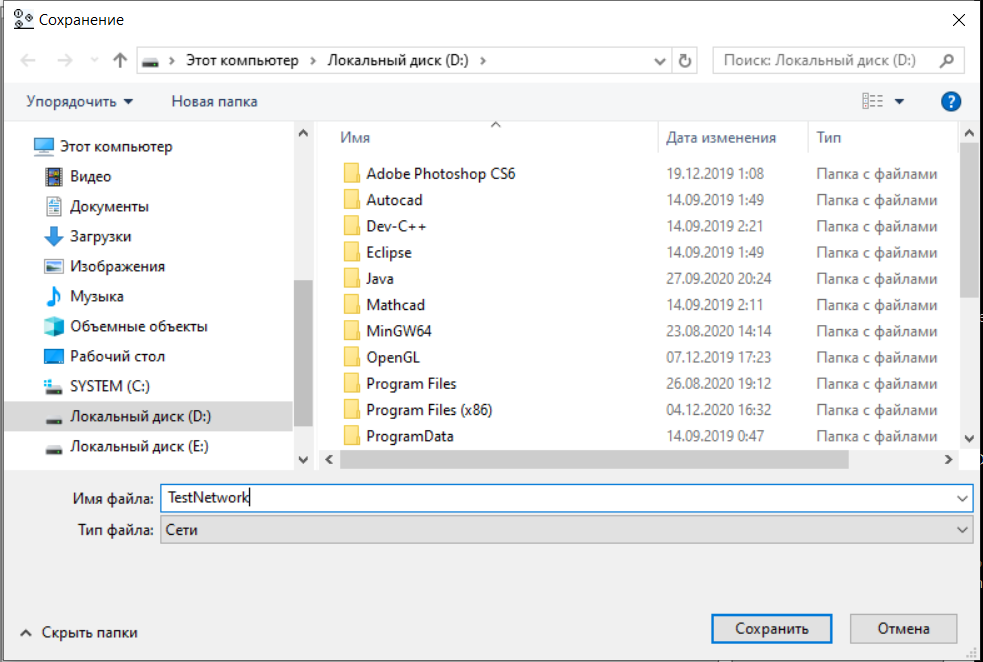


Рисунок 5.6

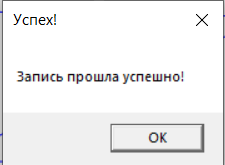


Рисунок 5.7

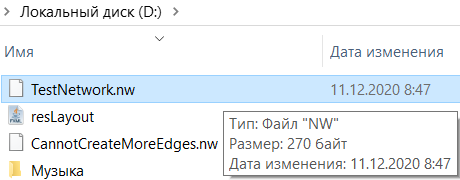


Рисунок 5.8

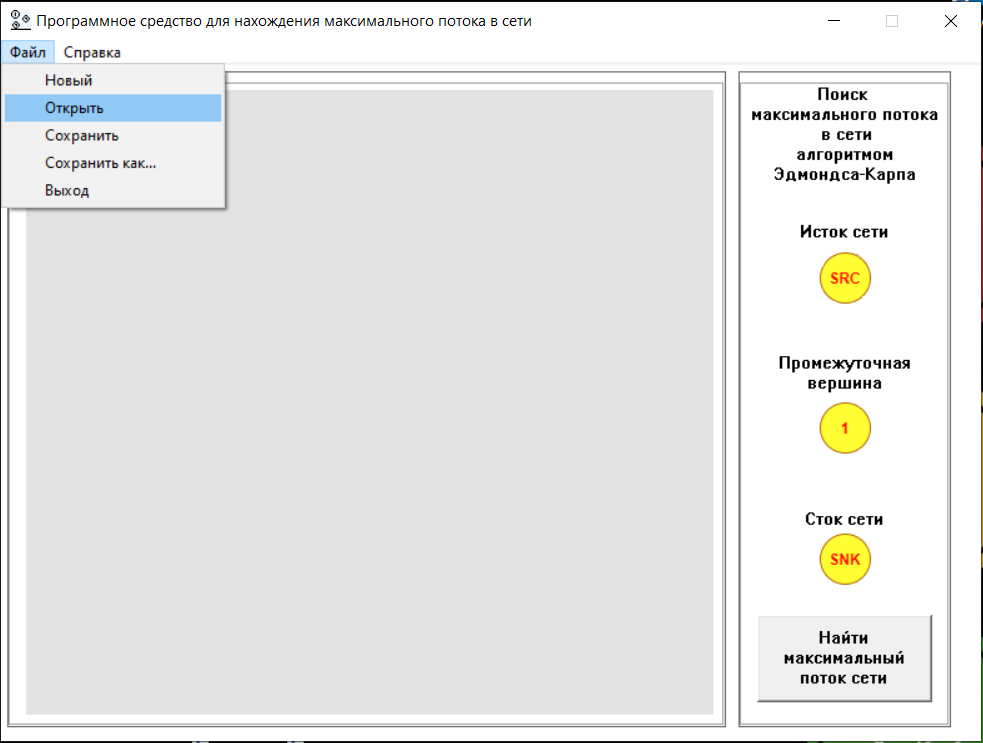


Рисунок 5.9

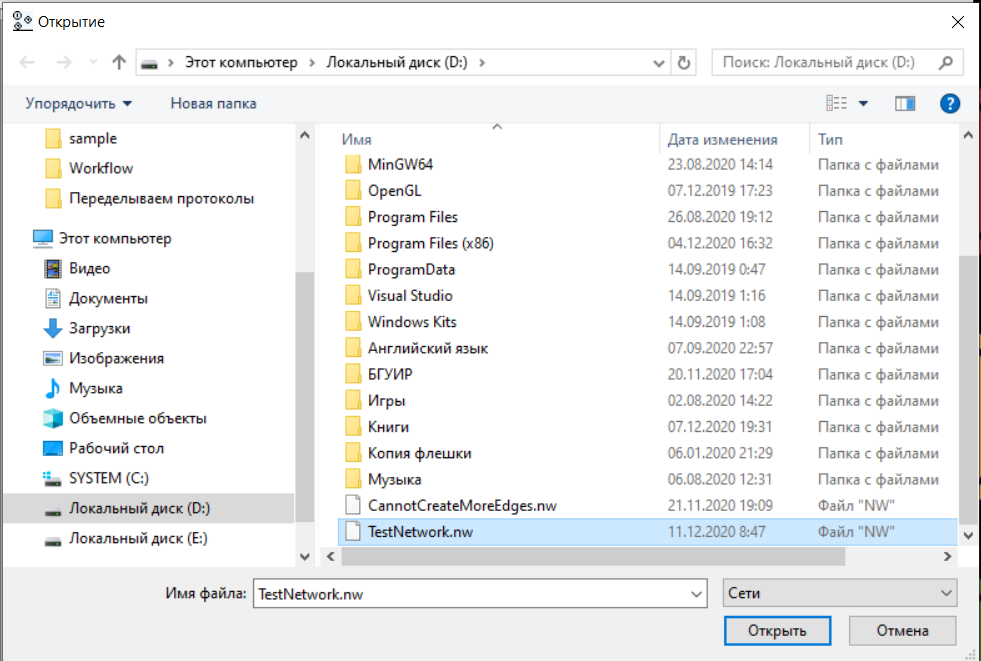


Рисунок 5.10

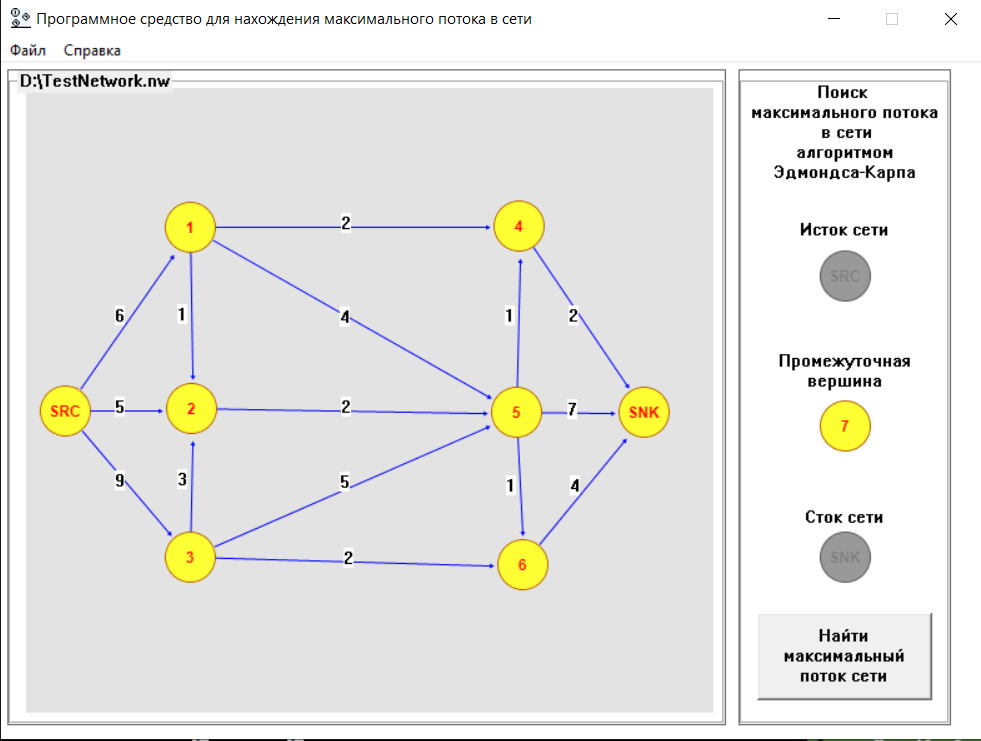


Рисунок 5.11

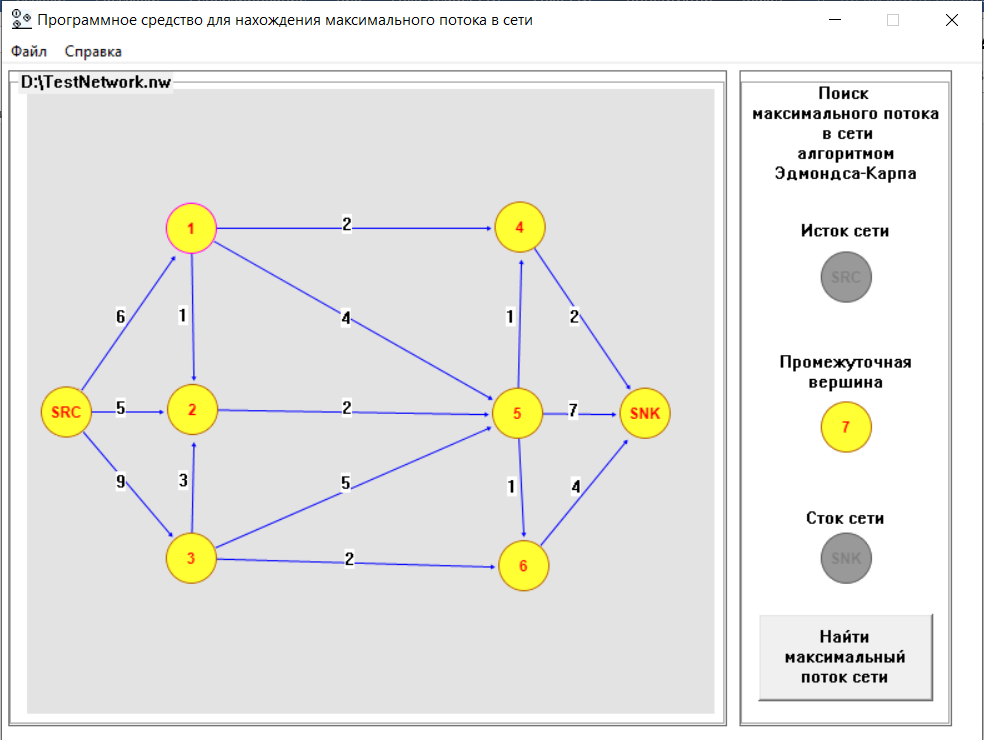


Рисунок 5.12

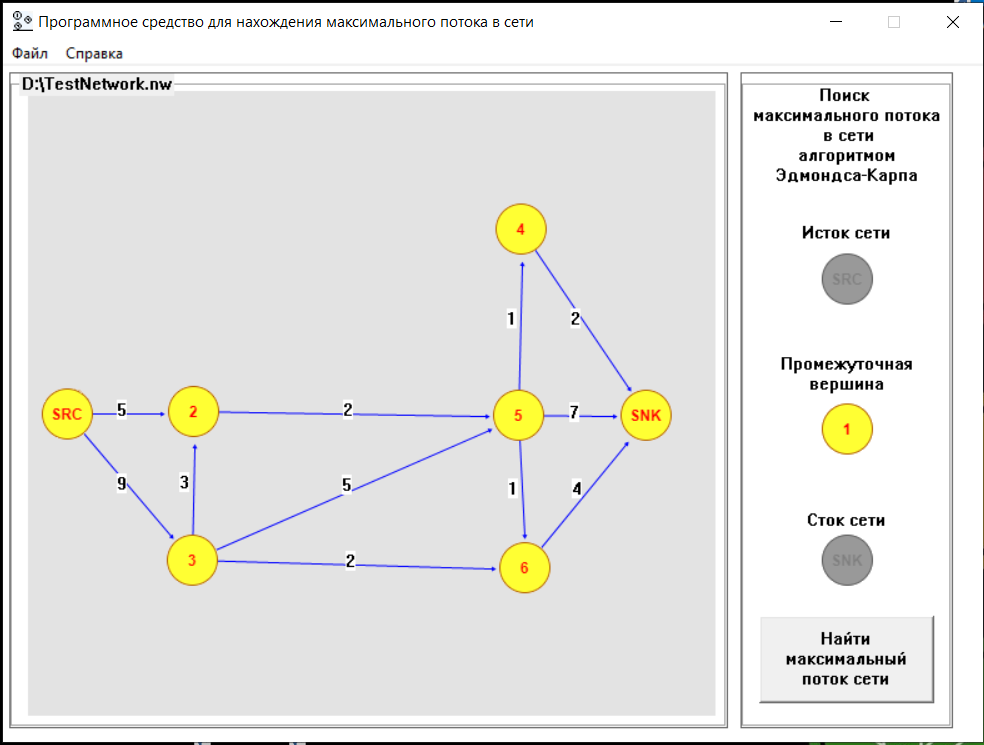


Рисунок 5.13

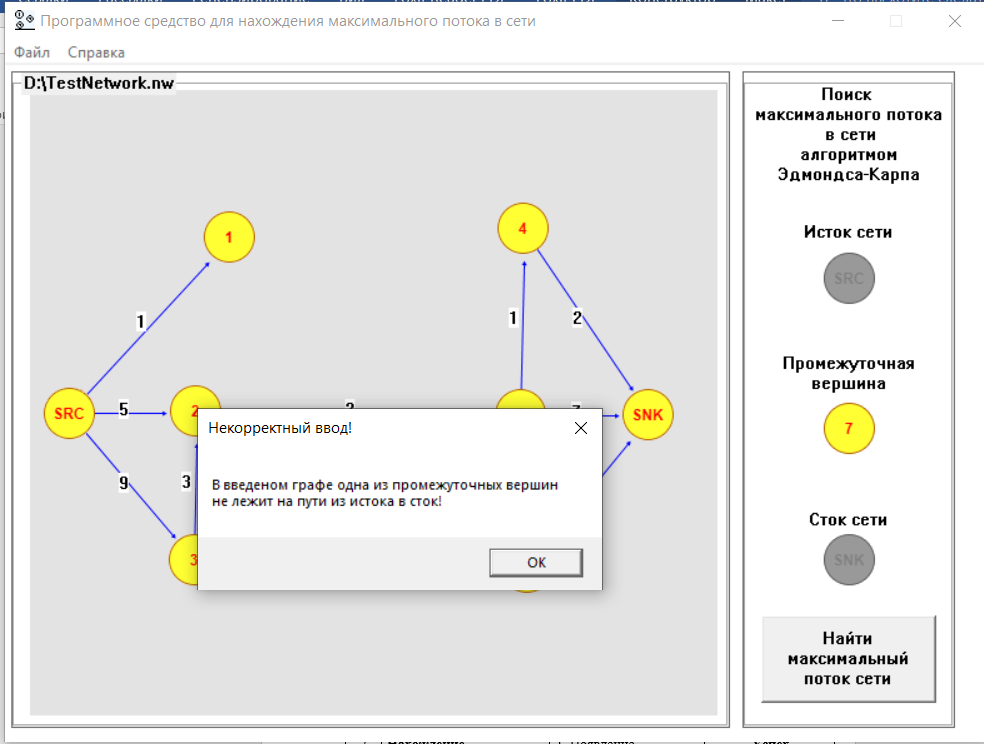


Рисунок 5.14

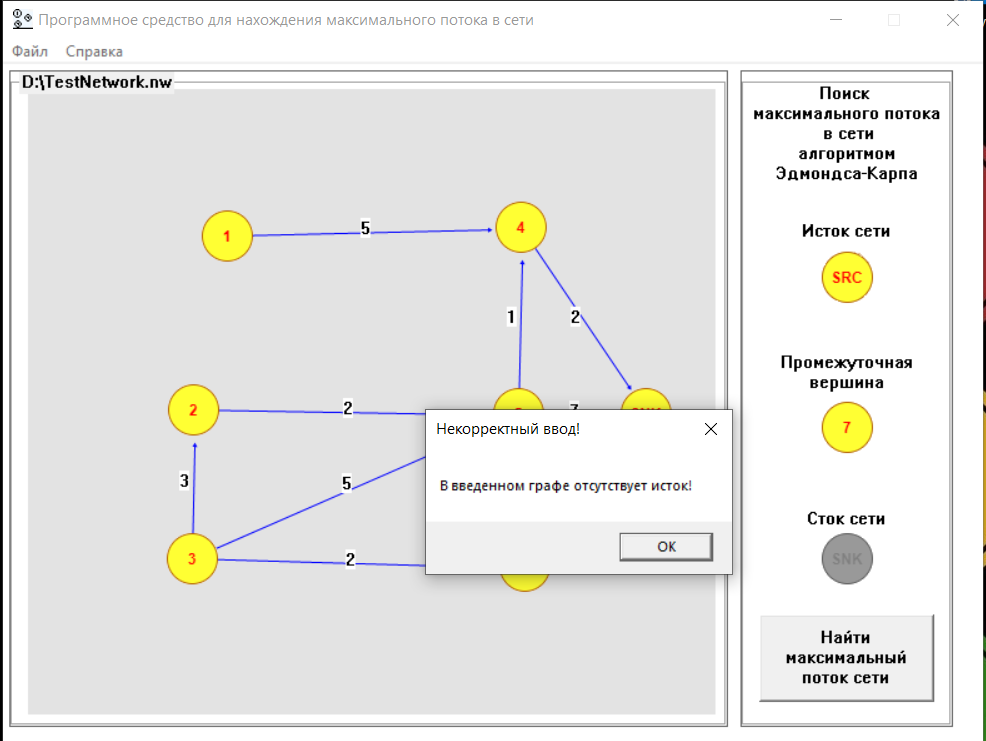


Рисунок 5.15

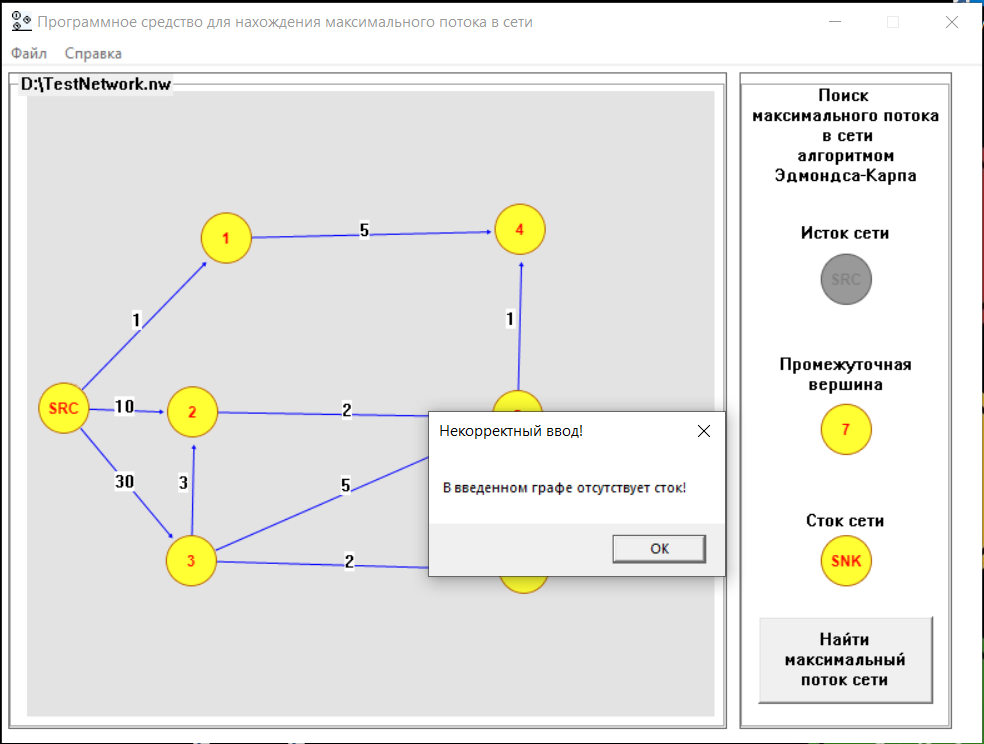


Рисунок 5.16

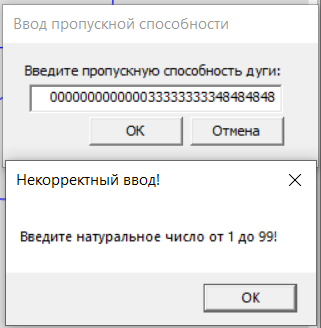


Рисунок 5.17

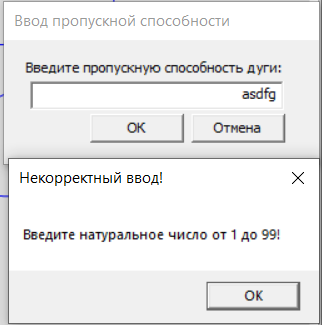


Рисунок 5.18

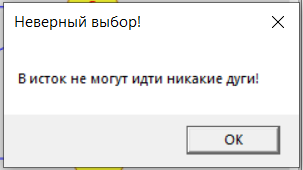


Рисунок 5.19

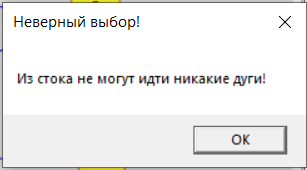


Рисунок 5.20

**6 РУКВОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для того, чтобы пользователь мог начать работать с приложением, ему необходимо иметь операционную систему Windows на своем компьютере версии XP и выше.

Если это требование выполнено, то пользователю достаточно запустить WinAPICourseProject.exe. После запуска пользователь увидит приложение, готовое к работе (рисунок 6.1).



Рисунок 6.1 – Стартовое окно

Для перенесения фишек-вершин сети достаточно нажать по одной из них, после чего курсор компьютера поменяется в перекрестие, затем перекрестием выбрать точку в поле вывода приложения (серое поле). Фишка окажется на положенном месте, а курсор вернется в обычное состояние. Если же будет выбрана точка, не принадлежащая полю, то перекрестие превратится в обычный курсор, а фишка вернется в свой слот. Возможное расположение фишек представлено на рисунке 6.2.

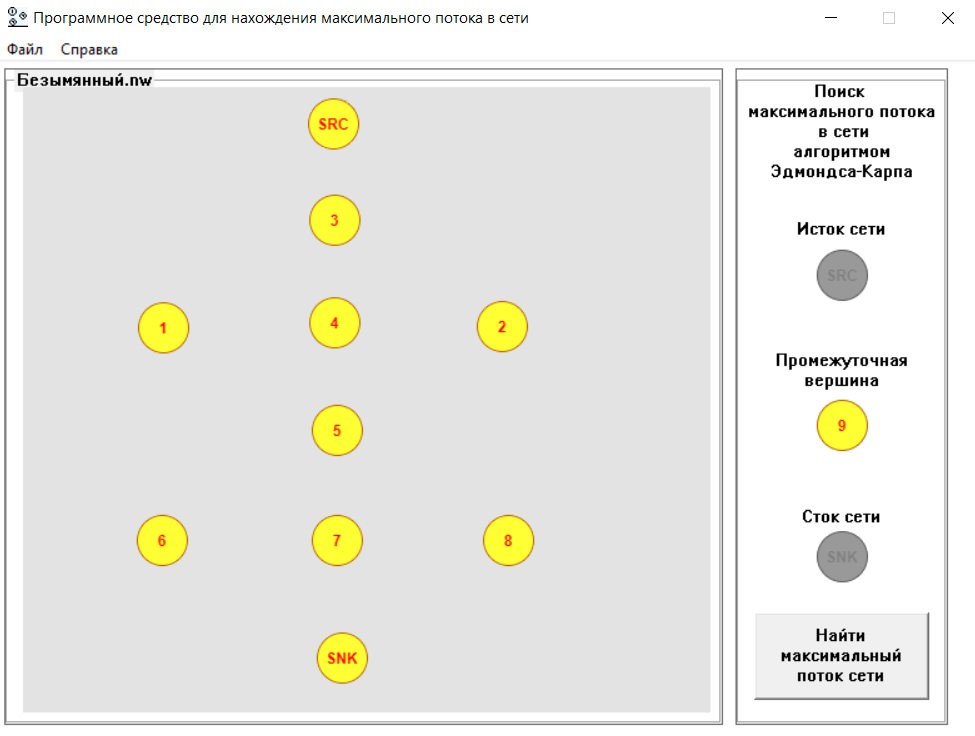


Рисунок 6.2 – Поле ввода с произвольным расположением фишек

Для создания дуг в сети пользователю достаточно выделить левой кнопкой мыши вершину, желаемую в качестве вершины, из которой будет выходить дуга. После выделения вершина приобретет розовую границу. После этого пользователю необходимо выбрать вершину, желаемую в качестве принимающей. После этого перед пользователем появится диалоговое окно для ввода пропускной способности дуги, представленное на рисунке 6.3.

Для ввода ожидается натуральное число, не превышающее 99. В случае некорректного ввода программа предупредит пользователя об этом.

После расставления пропускных способней всем дугам, сеть может выглядеть, как показано на рисунке 6.4.

Для нахождения максимального потока введенной сети пользователю достаточно нажать кнопку «Найти максимальный поток сети», после чего для обработки корректно введенной сети будет применен алгоритм Эдмондса-Карпа. По завершении работы алгоритма на экране будет представлено диалоговое окно с выводом сети с распределенными единицами потока по всем дугам, а также самое значение максимального потока сети., как на рисунке 6.5.

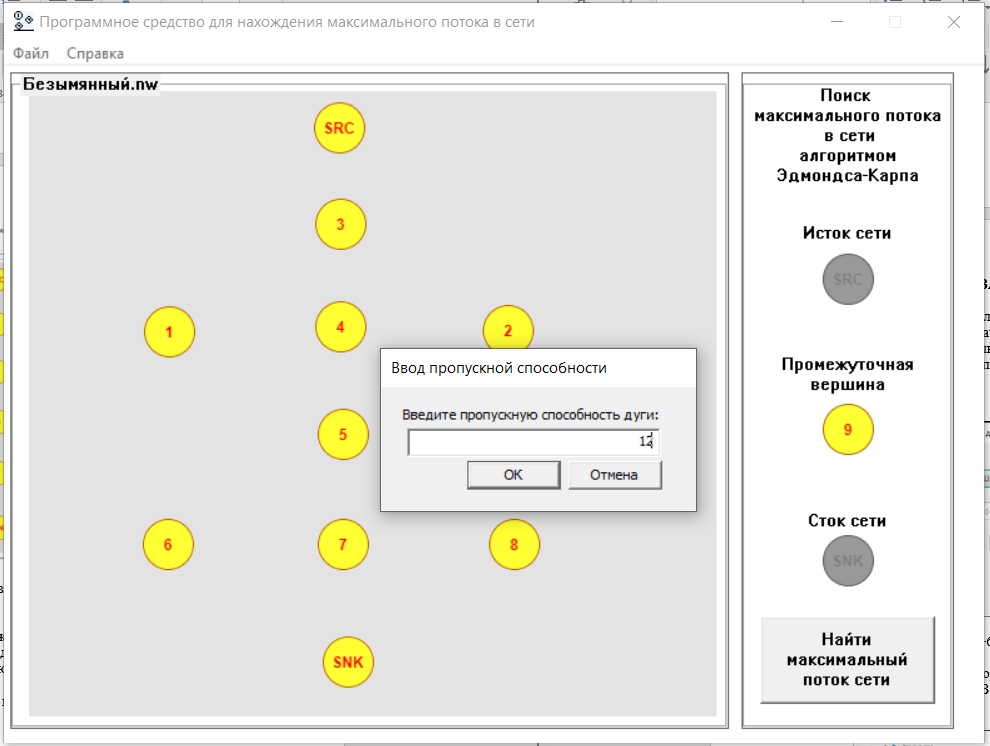


Рисунок 6.3 – Ввод пользователем пропускной способности дуги

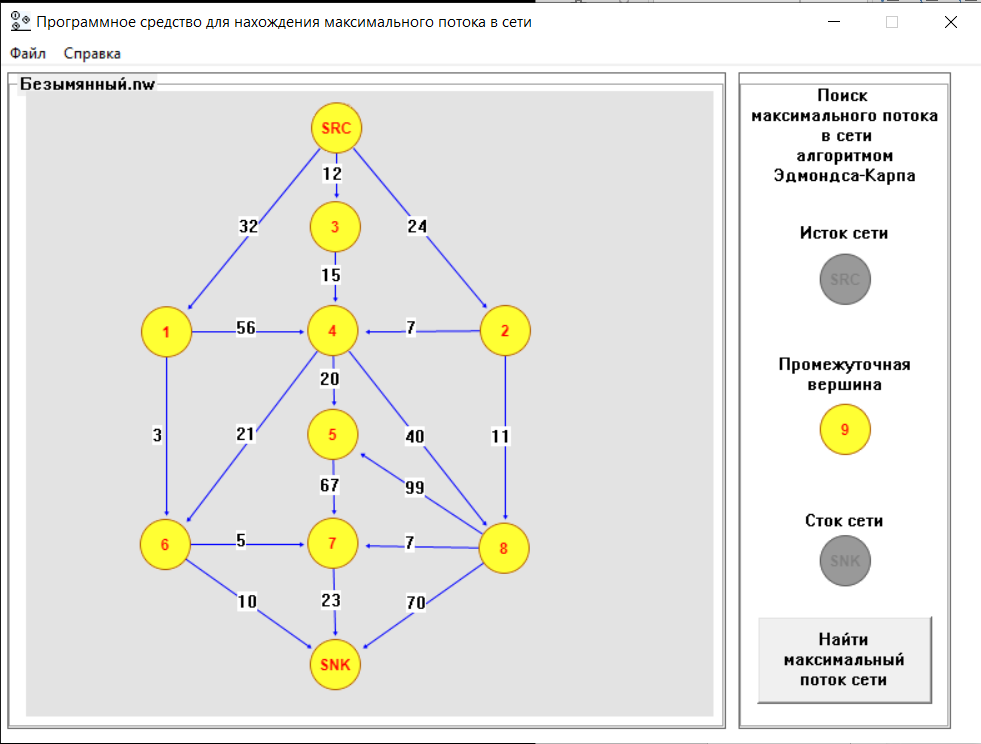


Рисунок 6.4 – Изображение готовой сети

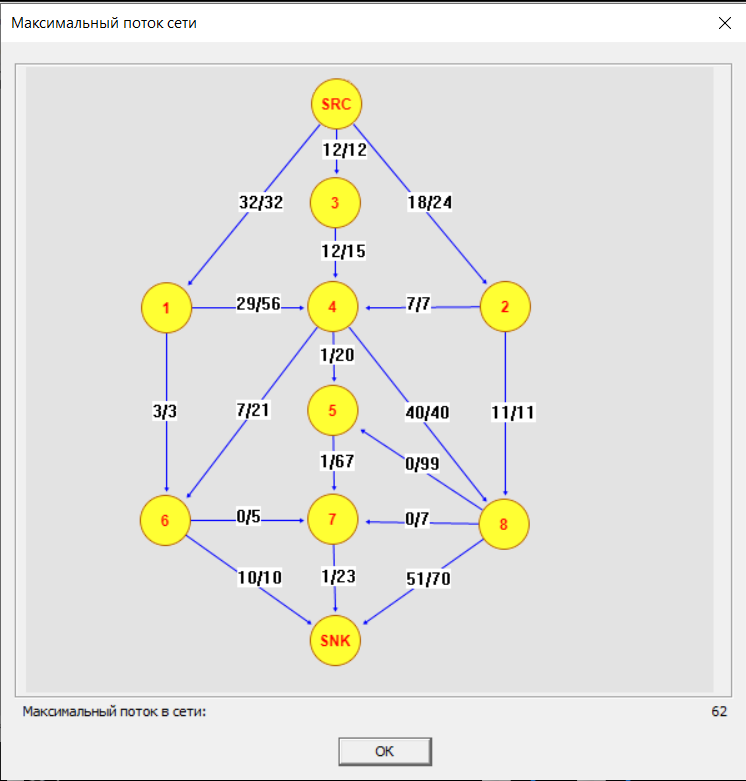


Рисунок 6.5 – Диалоговое окно с выводом результата работы программы

Если же пользователь ввёл некорректную сеть, то будет выдано соответствующее предупреждение.

Для удаление некоторой дуги из сети достаточно выделить конечную вершину, а затем нажать по вершине-источнику для конкретной дуги. Таким образом, дуга будет удалена из сети, как представлено на рисунках 6.6-6.8.

Для удаления некоторой вершины достаточно один раз нажать левой кнопкой мыши по ней, после чего нажать кнопку «Backspace» на клавиатуре. После этого вершина будет убрана из сети, а все входящие в нее и исходящие из нее ребра будут удалены, как показано на рисунках 6.9 и 6.10.

Также пользователь имеет возможность открывать готовые сети и сохранять новые. Для этого достаточно выбрать пункт «Файл» в строке меню приложения и выбрать соответствующие пункты подменю, как показано на рисунке 6.11. Процесс открытия и сохранения сопровождается уже знакомыми диалоговыми окнами, как показано на рисунках 6.12 и 6.13.

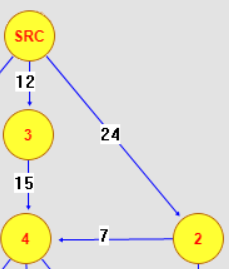


Рисунок 6.6 – Дуга, идущая из SRC в вершину 2

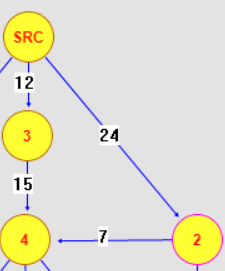


Рисунок 6.7 – Выделение вершины 2

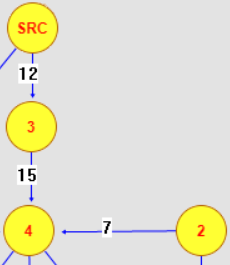


Рисунок 6.8 – Удаление дуги

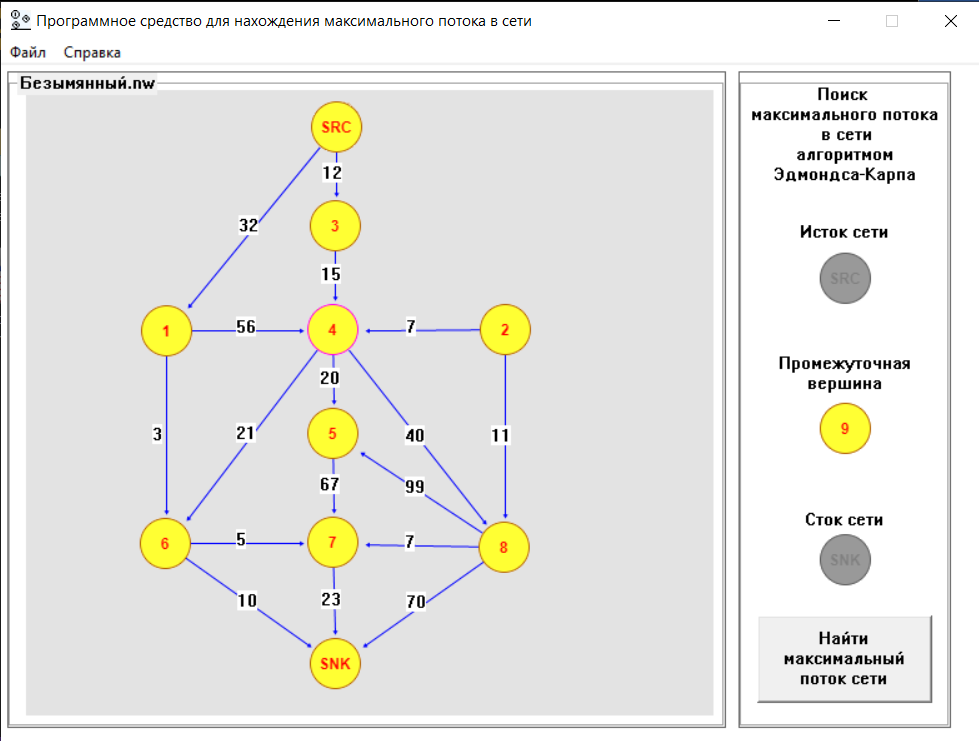


Рисунок 6.9 – В сети выделена вершина с номером 4

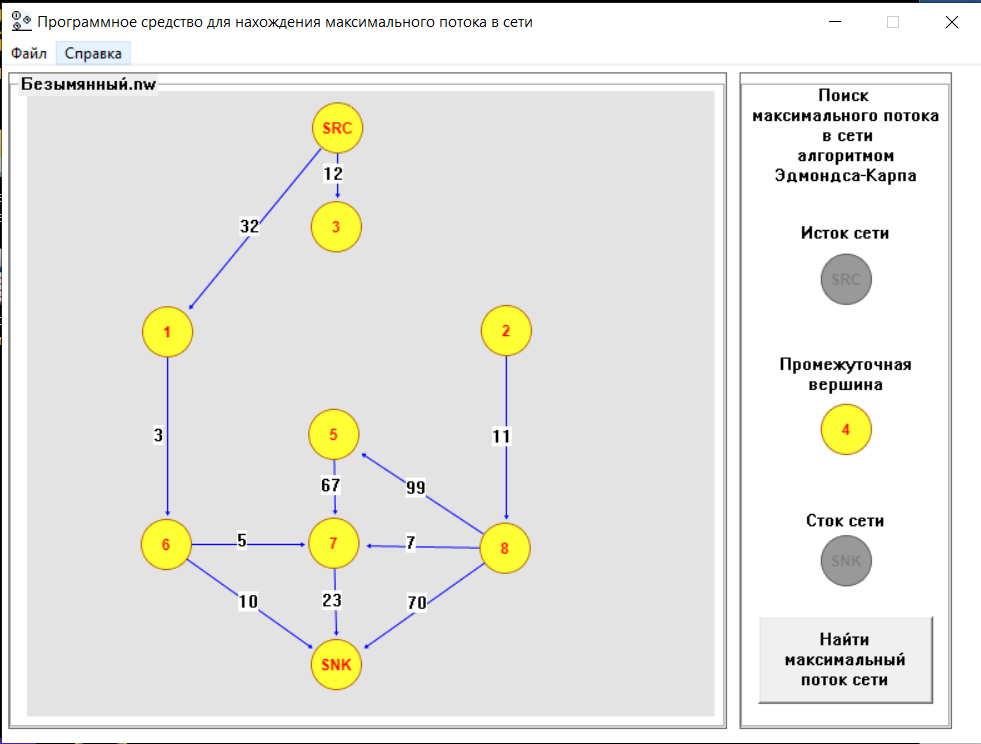


Рисунок 6.10 – Вершина 4 удалена после нажатия клавиши «Backspace»

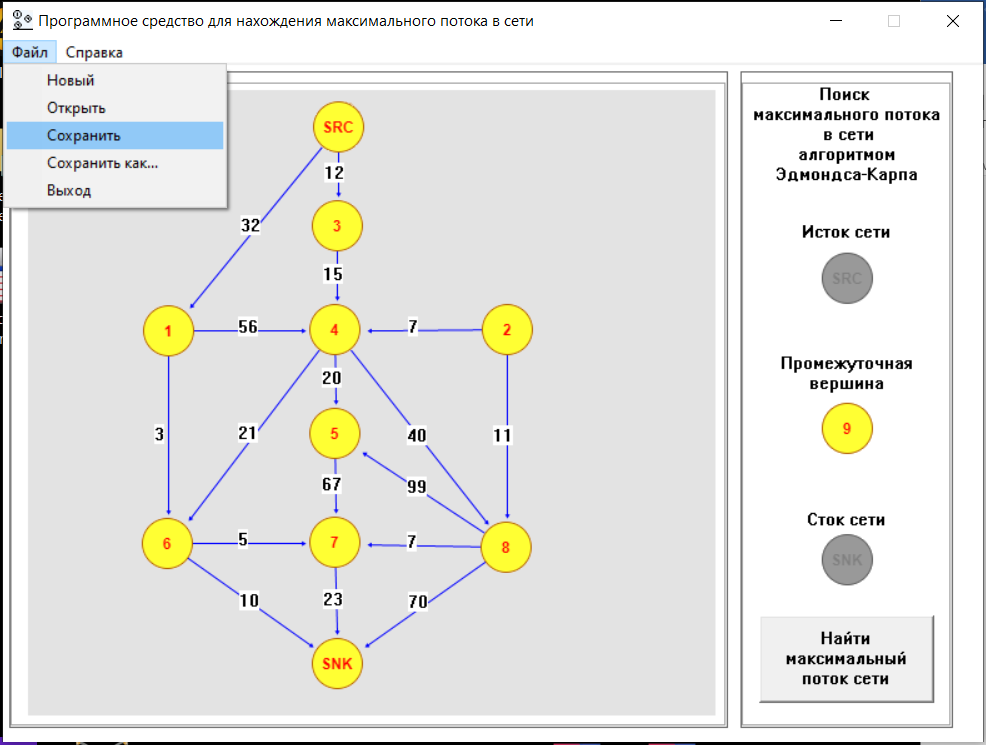


Рисунок 6.11 – Пункт меню «Файл»

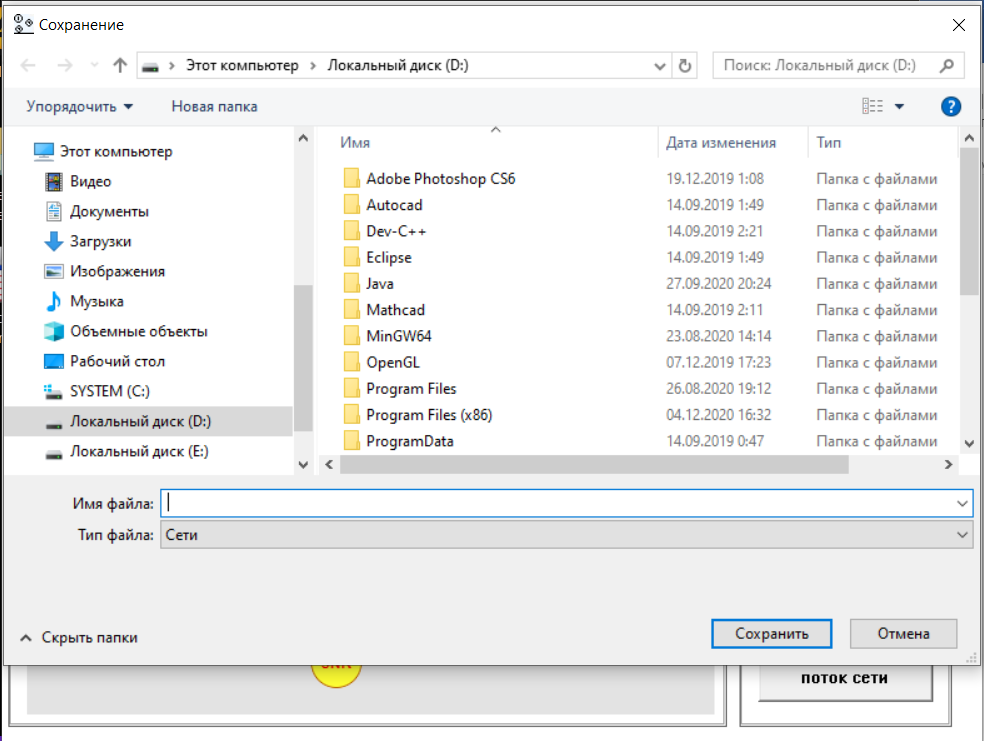


Рисунок 6.12 – Окно сохранения файла

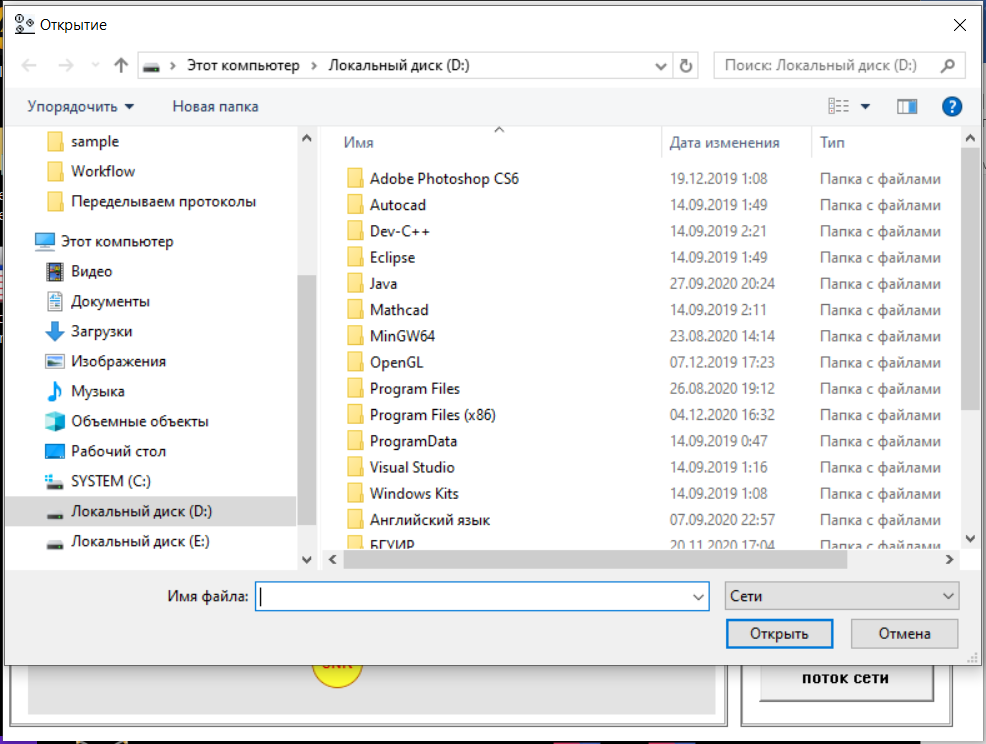


Рисунок 6.13 – Окно открытия файла

Для выхода из приложения достаточно нажать крестик в правом верхнем углу окна либо же выбрать соответствующий пункт подменю в вкладке «Файл».

**ЗАКЛЮЧЕНИЕ**

Результатом работы над курсовым проектом стало создание исправного и рабочего программного средства для нахождения максимального потока вводимой пользователем сети алгоритмом Эдмондса-Карпа.

Были детально изучены алгоритм и теорема Фалкерсона, алгоритм Дейкстры, а также сам алгоритм Эдмондса-Карпа, его алгоритмическая сложность, а также сферы применения.

Были изучены многие стороны и аспекты языка С/C++, его обширная библиотека стандартных функций, а также инструменты, при помощи которых можно создать GUI, используя С/С++. Наиболее глубоко были изучены возможности WinAPI.

Было проведено ознакомление с реализацией ООП в Windows/WinAPI.

Были тщательно продуманы и разработаны архитектура приложения, оперируемые в ней данные, а также все необходимые алгоритмы для получения требуемого результата.

Полученное приложение было разработано и многократно протестировано с соблюдением всех принципов разработки программного обеспечения.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Introduction to algorithms / Thomas H. Cormen . . . [et al.].—3rd ed – 1313 с.
2. Library of Congress Cataloging-in-Publication Data Sedgewick, Robert, 1946 Algorithms in C++ / Robert Sedgewick.—3d ed.- 528 c.
3. C++ Programming Robert Sedgewick - Princeton University Addison Wesley Professional Algorithms in C++, Parts 1A4: Fundamentals, Data Structure, Sorting, Searching, Third Edition – 752 с.
4. Керниган Б., Ритчи Д. Язык программирования Си ДПер. с англ., 3-е изд., испр. — СПб.: "Невский Диалект", 2001. - 352 с.
5. Library of Congress Cataloging-in-Publication Data Stroustrup,Bjarne The C++ Programming Language / Bjarne Stroustrup. — 3rd. ed. – 1022 с
6. Эккель Б. Философия С++. Введение в стандартный С++. 2-е изд. – СПб.: Питер, 2004. – 672 с.
7. Эккель Б., Эллисон Ч. Философия С++. Практическое программирование. – СПб: Питер, 2004. – 608 с.
8. Щупак Ю. А. Win32 API. Разработка приложений для Windows. – СПб.: Питер, 2008. – 529 с.
9. Домашняя страница документации и учебных ресурсов Майкрософт для разработчиков и технических специалистов [Электронный ресурс].

URL: <https://docs.microsoft.com/ru-ru/>

1. Дистанционная подготовка по информатике [Электронный ресурс].

URL: https://informatics.mccme.ru/mod/statements/view3.php?chapterid=2783#1

**ПРИЛОЖЕНИЕ. ИСХОДНЫЙ КОД ПРОГРАММЫ**

**EdmondKarpAlgorithm.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "EdmondKarpAlgorithm.h"

#define MAX\_ARR\_LENGTH 100

netEdge network[MAX\_ARR\_LENGTH][MAX\_ARR\_LENGTH];

int resNet[MAX\_ARR\_LENGTH][MAX\_ARR\_LENGTH];

const int MAX\_VERTEXES = 20;

void initializeNetworkWithZeros(void) {

for (int i = 0; i < MAX\_ARR\_LENGTH; i++)

for (int j = 0; j < MAX\_ARR\_LENGTH; j++) {

network[i][j].flow = 0;

network[i][j].capacity = 0;

}

}

void initializeResidualNetworkWithZeros(void) {

for (int i = 0; i < MAX\_ARR\_LENGTH; i++)

for (int j = 0; j < MAX\_ARR\_LENGTH; j++)

resNet[i][j] = 0;

}

int SINK\_VERTEX = 0;

int N = 0;

int M = 0;

void initializeNetworkWithData(int \*\*w) {

for (int i = 0; i < M; i++) {

int u = w[i][0], v = w[i][1], c = w[i][2];

u--;

v--;

network[u][v].capacity = c;

}

}

void handleAntiparallelEdges(void) {

for (int u = 0; u < N; u++) {

for (int v = 0; v < N; v++) {

if (network[u][v].capacity > 0 &&

network[v][u].capacity > 0) {

int c = network[v][u].capacity;

network[v][u].capacity = 0;

network[v][N].capacity = c;

network[N][u].capacity = c;

N++;

}

}

}

}

void updateResidualNetwork(void) {

for (int u = 0; u < N; u++) {

for (int v = 0; v < N; v++) {

int cuv = network[u][v].capacity;

if (cuv > 0) {

resNet[u][v] = cuv - network[u][v].flow;

}

else {

resNet[u][v] = network[v][u].flow;

}

}

}

}

short int visited[MAX\_ARR\_LENGTH];

int distance[MAX\_ARR\_LENGTH];

int shortest[MAX\_ARR\_LENGTH];

#define INFINITY 9999999

int DijkstraAlgorithm(int source, int sink) {

for (int i = 0; i < N; i++) {

distance[i] = INFINITY;

visited[i] = 0;

shortest[i] = -1;

}

distance[source] = 0;

for (int i = 0; i < N; i++) {

int min = INFINITY;

int u = 0;

for (int j = 0; j < N; j++)

if (!visited[j] && distance[j] < min) {

min = distance[j];

u = j;

}

visited[u] = 1;

for (int j = 0; j < N; j++)

if (!visited[j] && resNet[u][j] > 0 && distance[u] + 1 < distance[j]) {

distance[j] = distance[u] + 1;

shortest[j] = u;

}

}

if (distance[sink] == INFINITY) {

return 0;

}

else {

int minResCap = INFINITY;

int curr = sink, prev = sink;

while (shortest[curr] != -1) {

curr = shortest[curr];

minResCap = (minResCap > resNet[curr][prev]) ? resNet[curr][prev] : minResCap;

prev = curr;

}

return minResCap;

}

}

void updateNetwork(int minResCap) {

int curr = SINK\_VERTEX, prev = SINK\_VERTEX;

while (shortest[curr] != -1) {

curr = shortest[curr];

if (network[curr][prev].capacity > 0)

network[curr][prev].flow += minResCap;

else

network[prev][curr].flow -= minResCap;

prev = curr;

}

}

int checkForNetworkness(void) {

int flag = 0;

for (int i = 0; i < N; i++) {

if (network[0][i].capacity > 0) {

flag = 1;

break;

}

}

if (!flag) return -1;

flag = 0;

for (int i = 0; i < N; i++) {

if (network[i][N - 1].capacity > 0) {

flag = 1;

break;

}

}

if (!flag) return -2;

flag = 0;

for (int i = 1; i < N - 1; i++) {

int flag1 = 0;

for (int j = 0; j < N; j++) {

if (network[i][j].capacity > 0) {

flag1 = 1;

break;

}

}

int flag2 = 0;

for (int j = 0; j < N; j++) {

if (network[j][i].capacity > 0) {

flag2 = 1;

break;

}

}

if (!(flag1 && flag2)) {

flag = 1;

break;

}

}

if (flag) return -3;

return 0;

}

int EdmondKarpAlgorithmFunc(int \_N, int \_M, int \*\*w, netEdge (\*\*globPointer)[100]) {

initializeNetworkWithZeros();

initializeResidualNetworkWithZeros();

N = \_N;

M = \_M;

SINK\_VERTEX = N - 1;

int SOURCE\_VERTEX = 0;

initializeNetworkWithData(w);

int res = checkForNetworkness();

switch (res) {

case -1:

return -1;

break;

case -2:

return -2;

break;

case -3:

return -3;

break;

case 0:;

}

handleAntiparallelEdges();

updateResidualNetwork();

while (1) {

int minResCap = DijkstraAlgorithm(SOURCE\_VERTEX, SINK\_VERTEX);

if (minResCap == 0) break;

updateNetwork(minResCap);

updateResidualNetwork();

}

int MFLOW = 0;

for (int i = 0; i < N; i++)

MFLOW += network[i][SINK\_VERTEX].flow;

\*globPointer = network;

return MFLOW;

}

**Source.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <windowsx.h>

#include <stdio.h>

#include <wingdi.h>

#include <debugapi.h>

#include "resource.h"

#include <math.h>

#include <objidl.h>

#include <gdiplus.h>

#include "EdmondKarpAlgorithm.h"

using namespace Gdiplus;

#define DRAW\_AREA\_UPPER\_LEFT\_X 20

#define DRAW\_AREA\_UPPER\_LEFT\_Y 20

#define DRAW\_AREA\_LOWER\_RIGHT\_X 570

#define DRAW\_AREA\_LOWER\_RIGHT\_Y 520

#define DRAW\_AREA\_WIDTH 550

#define DRAW\_AREA\_HEIGHT 500

#define MENU\_AREA\_UPPER\_LEFT\_X 590

#define MENU\_AREA\_UPPER\_LEFT\_Y 5

#define MENU\_AREA\_WIDTH 170

#define MENU\_AREA\_HEIGHT 525

#define COLOR\_TO\_BE\_TRANSPARENT RGB(153, 255, 204)

#define SRCBUTTONID 13

#define SNKBUTTONID 14

#define MEDBUTTONID 15

#define MAXFLOWBUTTONID 16

///

/// ////////////////////////////////////////////////////

///

typedef struct tagVertex {

int medNumber;

RECT vRect;

} VERTEX;

#define VERTEX\_MAX 20

VERTEX vertices[VERTEX\_MAX];

int v\_ind = 0;

///

/// ////////////////////////////////////////////////////

///

typedef struct tagEdge {

VERTEX\* u;

VERTEX\* v;

int cap;

} EDGE;

#define EDGE\_MAX 100

EDGE edges[EDGE\_MAX];

int e\_ind = 0;

///

/// ////////////////////////////////////////////////////

///

short int SRC\_IS\_SELECTED = 0;

short int SRC\_CREATED = 0;

short int SNK\_IS\_SELECTED = 0;

short int SNK\_CREATED = 0;

short int MED\_IS\_SELECTED = 0;

short int MED\_CREATED = 0;

short int DRAG\_ACTIVE = 0;

static HBITMAP HSRCBitmap;

static HBITMAP HSRCMonoBitmap;

static HBITMAP HSelectedSRCBitmap;

static HBITMAP HSNKBitmap;

static HBITMAP HSNKMonoBitmap;

static HBITMAP HSelectedSNKBitmap;

#define MAX\_MED 18

static HBITMAP HBMPs[MAX\_MED];

static HBITMAP HBMPs\_Selected[MAX\_MED];

static HBITMAP HBMPs\_Mono[MAX\_MED];

static short int alreadyCreated[MAX\_MED];

VERTEX\* firstChosenP = NULL;

VERTEX\* secondChosenP = NULL;

VERTEX\* sinkP = NULL;

VERTEX\* sourceP = NULL;

short int CAP\_RETURN\_RES = 0;

int MAX\_FLOW\_RES = 0;

BOOL CALLBACK RequestCapacityDlgProc(HWND, UINT, WPARAM, LPARAM);

netEdge (\*globalPointerToNetwork)[100] = NULL;

//=================================================================================================================

wchar\_t currFileName[256] = L"Безымянный.nw";

wchar\_t saveFileName[256] = L"\0";

OPENFILENAME ofn;

OPENFILENAME sfn;

HWND groupButton;

//=================================================================================================================

int writeAllDataToFile(void) {

FILE\* file;

file = \_wfopen(saveFileName, L"w");

if (file == NULL) return -1;

fwprintf(file, L"%d\n", v\_ind);

for (int i = 0; i < v\_ind; i++) {

fwprintf(file, L"%d %d %d %d %d\n",

vertices[i].vRect.left,

vertices[i].vRect.top,

vertices[i].vRect.right,

vertices[i].vRect.bottom,

vertices[i].medNumber);

}

fwprintf(file, L"%d\n", e\_ind);

for (int i = 0; i < e\_ind; i++) {

fwprintf(file, L"%d %d %d\n", edges[i].u->medNumber, edges[i].v->medNumber, edges[i].cap);

}

fclose(file);

return 0;

}

int SaveAllDataToFile(void) {

FILE\* file;

file = \_wfopen(currFileName, L"w");

if (file == NULL) return -1;

fwprintf(file, L"%d\n", v\_ind);

for (int i = 0; i < v\_ind; i++) {

fwprintf(file, L"%d %d %d %d %d\n",

vertices[i].vRect.left,

vertices[i].vRect.top,

vertices[i].vRect.right,

vertices[i].vRect.bottom,

vertices[i].medNumber);

}

fwprintf(file, L"%d\n", e\_ind);

for (int i = 0; i < e\_ind; i++) {

fwprintf(file, L"%d %d %d\n", edges[i].u->medNumber, edges[i].v->medNumber, edges[i].cap);

}

fclose(file);

return 0;

}

void ResetEverything(void) {

wcscpy(currFileName, L"Безымянный.nw");

SetWindowText(groupButton, currFileName);

e\_ind = 0;

v\_ind = 0;

sinkP = NULL;

sourceP = NULL;

firstChosenP = NULL;

secondChosenP = NULL;

SNK\_IS\_SELECTED = 0; SNK\_CREATED = 0;

SRC\_IS\_SELECTED = 0; SRC\_CREATED = 0;

MED\_IS\_SELECTED = 0; MED\_CREATED = 0;

DRAG\_ACTIVE = 0;

for (int i = 0; i < MAX\_MED; i++)

alreadyCreated[i] = 0;

}

void processReadData(void) {

sinkP = NULL;

sourceP = NULL;

firstChosenP = NULL;

secondChosenP = NULL;

SNK\_IS\_SELECTED = 0; SNK\_CREATED = 0;

SRC\_IS\_SELECTED = 0; SRC\_CREATED = 0;

MED\_IS\_SELECTED = 0; MED\_CREATED = 0;

DRAG\_ACTIVE = 0;

for (int i = 0; i < MAX\_MED; i++)

alreadyCreated[i] = 0;

for (int i = 0; i < v\_ind; i++) {

if (vertices[i].medNumber == -1) {

sourceP = &vertices[i];

SRC\_CREATED = 1;

}

else if (vertices[i].medNumber == -2) {

sinkP = &vertices[i];

SNK\_CREATED = 1;

}

else {

alreadyCreated[vertices[i].medNumber] = 1;

}

}

}

int readAllDataFromFile(void) {

FILE\* file;

file = \_wfopen(currFileName, L"r");

if (file == NULL) return -1;

int buf;

buf = fwscanf(file, L"%d\n", &v\_ind);

if (buf != 1) {

ResetEverything();

return -1;

}

for (int i = 0; i < v\_ind; i++) {

buf = fwscanf(file, L"%d %d %d %d %d\n",

&vertices[i].vRect.left,

&vertices[i].vRect.top,

&vertices[i].vRect.right,

&vertices[i].vRect.bottom,

&vertices[i].medNumber);

if (buf != 5) {

ResetEverything();

return -1;

}

}

buf = fwscanf(file, L"%d\n", &e\_ind);

if (buf != 1) {

ResetEverything();

return -1;

}

for (int i = 0; i < e\_ind; i++) {

int u, v, cap;

buf = fwscanf(file, L"%d %d %d\n", &u, &v, &cap);

edges[i].u = NULL;

edges[i].v = NULL;

for (int j = 0; j < v\_ind; j++)

if (vertices[j].medNumber == u) {

edges[i].u = &vertices[j];

break;

}

if (edges[i].u == NULL) {

ResetEverything();

return -1;

}

for (int j = 0; j < v\_ind; j++)

if (vertices[j].medNumber == v) {

edges[i].v = &vertices[j];

break;

}

if (edges[i].v == NULL) {

ResetEverything();

return -1;

}

edges[i].cap = cap;

if (buf != 3) {

ResetEverything();

return -1;

}

}

fclose(file);

processReadData();

return 0;

}

void printCurrentStateDebug(void) {

OutputDebugStringA("\n\n===============================================================================\n\n");

char str[256];

sprintf(str, "TOTAL NUMBER OF VERTICES: %d\n", v\_ind);

OutputDebugStringA(str);

for (int i = 0; i < v\_ind; i++) {

sprintf(str, "\nARRAY INDEX: %d\nvMEDNUMBER: %d\n", i, vertices[i].medNumber);

OutputDebugStringA(str);

if (&vertices[i] == sourceP) OutputDebugStringA("IT'S A SOURCE!!!!!!!\n\n");

if (&vertices[i] == sinkP) OutputDebugStringA("IT'S A SINK!!!!!!!\n\n");

}

OutputDebugStringA("\n\n===============================================================================\n\n");

}

int getGreatestVertexNumber(void) {

int greatest = -1;

for (int i = 0; i < v\_ind; i++)

if (vertices[i].medNumber > greatest)

greatest = vertices[i].medNumber;

return greatest;

}

void printCurrentStateDebugDifferent(void) {

OutputDebugStringA("\n\n===============================================================================\n\n");

char str[256];

sprintf(str, "TOTAL NUMBER OF VERTICES: %d\n", v\_ind);

OutputDebugStringA(str);

sprintf(str, "TOTAL NUMBER OF EDGES: %d\n", e\_ind);

OutputDebugStringA(str);

int greatest = getGreatestVertexNumber() + 2;

for (int i = 0; i < e\_ind; i++) {

sprintf(str, "\nFrom: %d to: %d with %d\n", (edges[i].u->medNumber == -2) ? greatest : edges[i].u->medNumber + 1, (edges[i].v->medNumber == -2) ? greatest : edges[i].v->medNumber + 1, edges[i].cap);

OutputDebugStringA(str);

}

OutputDebugStringA("\n\n===============================================================================\n\n");

}

void RedrawDrawAreaBackground(HDC hMemDC) {

RECT rect;

SetRect(&rect, 0, 0, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT);

FillRect(hMemDC, &rect, (HBRUSH)(COLOR\_3DLIGHT + 1));

}

int getNotCreatedMedIndex(void);

int CreateVertex(int x, int y) {

if (v\_ind == VERTEX\_MAX) {

return -VERTEX\_MAX;

}

else {

if (SRC\_IS\_SELECTED) {

if (SRC\_CREATED) {

return -1;

}

else {

vertices[v\_ind].medNumber = -1;

sourceP = &vertices[v\_ind];

SRC\_CREATED = 1;

}

}

if (SNK\_IS\_SELECTED) {

if (SNK\_CREATED) {

return -2;

}

else {

vertices[v\_ind].medNumber = -2;

sinkP = &vertices[v\_ind];

SNK\_CREATED = 1;

}

}

if (MED\_IS\_SELECTED) {

int res = getNotCreatedMedIndex();

if (res == MAX\_MED) {

return -3;

}

else {

vertices[v\_ind].medNumber = res;

MED\_CREATED = 1;

}

}

if (SRC\_IS\_SELECTED || SNK\_IS\_SELECTED || MED\_IS\_SELECTED) {

SetRect(&vertices[v\_ind].vRect, x - 21, y - 21, x + 21, y + 21);

v\_ind++;

return 0;

}

else {

return -5;

}

}

}

int CreateEdge(VERTEX \*u, VERTEX \*v, int capacity) {

if (e\_ind == EDGE\_MAX) {

return -EDGE\_MAX;

}

else {

edges[e\_ind].u = u;

edges[e\_ind].v = v;

edges[e\_ind].cap = capacity;

e\_ind++;

return 0;

}

}

EDGE \*EdgeExists(VERTEX\* u, VERTEX\* v) {

for (int i = 0; i < e\_ind; i++)

if (edges[i].u == u && edges[i].v == v) return &edges[i];

return NULL;

}

void DrawVertex(HDC hdc, VERTEX\* v) {

HDC hMemDC = CreateCompatibleDC(hdc);

int x = v->vRect.left;

int y = v->vRect.top;

if (v == sourceP)

SelectObject(hMemDC, HSRCBitmap);

else if (v == sinkP)

SelectObject(hMemDC, HSNKBitmap);

else {

SelectObject(hMemDC, HBMPs[v->medNumber]);

MED\_CREATED = 0;

alreadyCreated[v->medNumber] = 1;

}

TransparentBlt(hdc, x, y, 41, 41,

hMemDC, 0, 0, 41, 41, COLOR\_TO\_BE\_TRANSPARENT);

DeleteDC(hMemDC);

}

void DrawSelectedVertex(HDC hdc, VERTEX\* v) {

HDC hMemDC = CreateCompatibleDC(hdc);

int x = v->vRect.left;

int y = v->vRect.top;

if (v == sourceP)

SelectObject(hMemDC, HSelectedSRCBitmap);

else if (v == sinkP)

SelectObject(hMemDC, HSelectedSNKBitmap);

else

SelectObject(hMemDC, HBMPs\_Selected[v->medNumber]);

TransparentBlt(hdc, x, y, 41, 41,

hMemDC, 0, 0, 41, 41, COLOR\_TO\_BE\_TRANSPARENT);

DeleteDC(hMemDC);

}

void DrawAllVerticesBack(HDC hMemDC) {

HDC diffHDC = CreateCompatibleDC(hMemDC);

for (int i = 0; i < v\_ind; i++) {

int x = vertices[i].vRect.left;

int y = vertices[i].vRect.top;

if (firstChosenP == &vertices[i]) {

if (&vertices[i] == sourceP)

SelectObject(diffHDC, HSelectedSRCBitmap);

else if (&vertices[i] == sinkP)

SelectObject(diffHDC, HSelectedSNKBitmap);

else

SelectObject(diffHDC, HBMPs\_Selected[vertices[i].medNumber]);

}

else {

if (&vertices[i] == sourceP)

SelectObject(diffHDC, HSRCBitmap);

else if (&vertices[i] == sinkP)

SelectObject(diffHDC, HSNKBitmap);

else

SelectObject(diffHDC, HBMPs[vertices[i].medNumber]);

}

TransparentBlt(hMemDC, x - 20, y - 20, 41, 41,

diffHDC, 0, 0, 41, 41, COLOR\_TO\_BE\_TRANSPARENT);

}

DeleteDC(diffHDC);

}

double getVectorLength(double x, double y) {

return sqrt(x \* x + y \* y);

}

void DrawAllEdgesBack(HDC hMemDC) {

for (int i = 0; i < e\_ind; i++) {

int upperLeftX = edges[i].u->vRect.left;

int upperLeftY = edges[i].u->vRect.top;

int x1 = upperLeftX + 20;

int y1 = upperLeftY + 20;

int lowerRightX = edges[i].v->vRect.right;

int lowerRightY = edges[i].v->vRect.bottom;

int x2 = lowerRightX - 20;

int y2 = lowerRightY - 20;

double origLength = getVectorLength((double) x2 - x1, (double)y2 - y1);

double newLength = origLength - 25;

double k = newLength / origLength;

int newX2 = (int) floor(x1 + ((double)x2 - x1) \* k);

int newY2 = (int) floor(y1 + ((double)y2 - y1) \* k);

Graphics graphics(hMemDC);

graphics.SetSmoothingMode(SmoothingModeAntiAlias);

Pen pen(Color(255, 0, 0, 255));

pen.SetEndCap(LineCapArrowAnchor);

graphics.DrawLine(&pen, x1 - 20, y1 - 20, newX2 - 20, newY2 - 20);

RECT rect;

SetRect(&rect, (x1 + x2) / 2 - 11 - 20, (y1 + y2) / 2 - 11 - 20, (x1 + x2) / 2 + 11 - 20, (y1 + y2) / 2 + 11 - 20);

char str[256];

\_itoa(edges[i].cap, str, 10);

DrawTextA(hMemDC, str, -1, &rect, (UINT)NULL);

}

}

void DrawAllEdgesBackToRes(HDC hMemDC) {

for (int i = 0; i < e\_ind; i++) {

int upperLeftX = edges[i].u->vRect.left;

int upperLeftY = edges[i].u->vRect.top;

int x1 = upperLeftX + 20;

int y1 = upperLeftY + 20;

int lowerRightX = edges[i].v->vRect.right;

int lowerRightY = edges[i].v->vRect.bottom;

int x2 = lowerRightX - 20;

int y2 = lowerRightY - 20;

double origLength = getVectorLength((double)x2 - x1, (double)y2 - y1);

double newLength = origLength - 25;

double k = newLength / origLength;

int newX2 = (int)floor(x1 + ((double)x2 - x1) \* k);

int newY2 = (int)floor(y1 + ((double)y2 - y1) \* k);

Graphics graphics(hMemDC);

graphics.SetSmoothingMode(SmoothingModeAntiAlias);

Pen pen(Color(255, 0, 0, 255));

pen.SetEndCap(LineCapArrowAnchor);

graphics.DrawLine(&pen, x1 - 20, y1 - 20, newX2 - 20, newY2 - 20);

RECT rect;

SetRect(&rect, (x1 + x2) / 2 - 11 - 20, (y1 + y2) / 2 - 11 - 20, (x1 + x2) / 2 + 31 - 20, (y1 + y2) / 2 + 11 - 20);

char str[256];

int u = edges[i].u->medNumber;

int v = edges[i].v->medNumber;

if (u == -1) u = 0;

else u++;

if (v == -2) v = getGreatestVertexNumber() + 2;

else v++;

\_itoa(globalPointerToNetwork[u][v].flow, str, 10);

char str2[256];

\_itoa(globalPointerToNetwork[u][v].capacity, str2, 10);

strcat(str, "/");

strcat(str, str2);

DrawTextA(hMemDC, str, -1, &rect, (UINT)NULL);

}

}

void DrawEverythingBack(HDC hdc) {

HDC hMemDC = CreateCompatibleDC(hdc);

HBITMAP memBM = CreateCompatibleBitmap(hdc, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT);

SelectObject(hMemDC, memBM);

RedrawDrawAreaBackground(hMemDC);

DrawAllEdgesBack(hMemDC);

DrawAllVerticesBack(hMemDC);

BitBlt(hdc, DRAW\_AREA\_UPPER\_LEFT\_X, DRAW\_AREA\_UPPER\_LEFT\_Y, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT, hMemDC, 0, 0, SRCCOPY);

DeleteDC(hMemDC);

DeleteObject(memBM);

}

void DrawEverythingBackToRes(HDC hdc) {

HDC hMemDC = CreateCompatibleDC(hdc);

HBITMAP memBM = CreateCompatibleBitmap(hdc, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT);

SelectObject(hMemDC, memBM);

RedrawDrawAreaBackground(hMemDC);

DrawAllEdgesBackToRes(hMemDC);

DrawAllVerticesBack(hMemDC);

BitBlt(hdc, DRAW\_AREA\_UPPER\_LEFT\_X, DRAW\_AREA\_UPPER\_LEFT\_Y, DRAW\_AREA\_WIDTH, DRAW\_AREA\_HEIGHT, hMemDC, 0, 0, SRCCOPY);

DeleteDC(hMemDC);

DeleteObject(memBM);

}

void DeselectButton(HWND hWnd, short int\* btn) {

HCURSOR hCurs = LoadCursor(NULL, IDC\_ARROW);

SetClassLongPtr(hWnd, GCLP\_HCURSOR, (LONG)(hCurs));

\*btn = 0;

}

void DeselectButtonsIfAny(HWND hWnd) {

if (SRC\_IS\_SELECTED) DeselectButton(hWnd, &SRC\_IS\_SELECTED);

if (SNK\_IS\_SELECTED) DeselectButton(hWnd, &SNK\_IS\_SELECTED);

if (MED\_IS\_SELECTED) DeselectButton(hWnd, &MED\_IS\_SELECTED);

InvalidateRect(hWnd, NULL, TRUE);

}

void RemoveEdgesFromArray(VERTEX\* p) {

while (1) {

int old\_ind = e\_ind;

for (int i = 0; i < e\_ind; i++)

if (edges[i].u == p || edges[i].v == p) {

for (int t = i + 1; t < e\_ind; t++)

edges[t - 1] = edges[t];

e\_ind--;

break;

}

if (old\_ind == e\_ind) break;

}

}

void AdjustVertexPointersInEdges(VERTEX\* p) {

for (int i = 0; i < e\_ind; i++) {

if (edges[i].u > p) edges[i].u--;

if (edges[i].v > p) edges[i].v--;

}

}

void DeleteVertexFromArray(HDC hdc, VERTEX\* p) {

if (p == sourceP) {

SRC\_CREATED = 0;

SRC\_IS\_SELECTED = 0;

if (sinkP != NULL) {

if (sinkP > sourceP) sinkP--;

}

sourceP = NULL;

}

else if (p == sinkP) {

SNK\_CREATED = 0;

SNK\_IS\_SELECTED = 0;

if (sourceP != NULL) {

if (sourceP > sinkP) sourceP--;

}

sinkP = NULL;

}

else {

MED\_CREATED = 0;

MED\_IS\_SELECTED = 0;

alreadyCreated[p->medNumber] = 0;

if (sourceP != NULL) {

if (sourceP > p) sourceP--;

}

if (sinkP != NULL) {

if (sinkP > p) sinkP--;

}

}

RemoveEdgesFromArray(p);

for (int i = 0; i < v\_ind; i++) {

if (&vertices[i] == p) {

int ind = i;

for (int j = ind + 1; j < v\_ind; j++) {

vertices[j - 1] = vertices[j];

}

v\_ind--;

break;

}

}

AdjustVertexPointersInEdges(p);

//DrawAllVerticesBack(hdc);

DrawEverythingBack(hdc);

}

void DeleteEdgeFromArray(HDC hdc, EDGE\* p) {

for (int i = 0; i < e\_ind; i++) {

if (&edges[i] == p) {

int ind = i;

for (int j = ind + 1; j < e\_ind; j++) {

edges[j - 1] = edges[j];

}

e\_ind--;

break;

}

}

DrawEverythingBack(hdc);

}

void ProcessClickInDrawArea(HWND hWnd, HDC hdc, int x, int y) {

if (SRC\_IS\_SELECTED || SNK\_IS\_SELECTED || MED\_IS\_SELECTED) {

int res = CreateVertex(x, y);

if (res == -VERTEX\_MAX) {

MessageBoxA(hWnd, "2 many", "Wrong!", (UINT)NULL);

}

else if (res == -1) {

MessageBoxA(hWnd, "SRC has been created already", "Wrong!", (UINT)NULL);

}

else if (res == -2) {

MessageBoxA(hWnd, "SNK has been created already", "Wrong!", (UINT)NULL);

}

else if (res == -3) {

MessageBoxA(hWnd, "2 many med", "Wrong!", (UINT)NULL);

}

else if (res == 0) {

DrawVertex(hdc, &vertices[v\_ind - 1]);

}

DeselectButtonsIfAny(hWnd);

}

else {

RECT rect;

HRGN hRgn;

for (int i = 0; i < v\_ind; i++) {

SetRect(&rect, vertices[i].vRect.left, vertices[i].vRect.top, vertices[i].vRect.right, vertices[i].vRect.bottom);

hRgn = CreateRectRgnIndirect(&rect);

if (PtInRegion(hRgn, x, y)) {

DRAG\_ACTIVE = 1;

if (firstChosenP == NULL) {

firstChosenP = &vertices[i];

DrawSelectedVertex(hdc, firstChosenP);

}

else

if (secondChosenP == NULL) {

secondChosenP = &vertices[i];

if (secondChosenP == firstChosenP) {

DrawVertex(hdc, firstChosenP);

firstChosenP = NULL;

secondChosenP = NULL;

}

else {

DrawVertex(hdc, firstChosenP);

if (secondChosenP == sourceP && !EdgeExists(secondChosenP, firstChosenP)) {

MessageBoxW(hWnd, L"В исток не могут идти никакие дуги!", L"Неверный выбор!", (UINT)NULL);

}

else if (firstChosenP == sinkP && !EdgeExists(secondChosenP, firstChosenP)) {

MessageBoxW(hWnd, L"Из стока не могут идти никакие дуги!", L"Неверный выбор!", (UINT)NULL);

}

else {

EDGE\* pointer = EdgeExists(secondChosenP, firstChosenP);

if (pointer != NULL) {

DeleteEdgeFromArray(hdc, pointer);

}

else {

DialogBox(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD\_REQUESTCAPACITY), hWnd, RequestCapacityDlgProc);

if (CAP\_RETURN\_RES >= 1 && CAP\_RETURN\_RES <= 99) {

pointer = EdgeExists(firstChosenP, secondChosenP);

if (pointer != NULL) {

pointer->cap = CAP\_RETURN\_RES;

//DrawEverythingBack(hdc);

}

else {

int res = CreateEdge(firstChosenP, secondChosenP, CAP\_RETURN\_RES);

if (res == -EDGE\_MAX) {

MessageBoxW(hWnd, L"Невозможно создать больше дуг!", L"Ошибка!", (UINT)NULL);

}

else {

//DrawEverythingBack(hdc);

}

}

}

}

}

firstChosenP = NULL;

secondChosenP = NULL;

DrawEverythingBack(hdc);

}

}

DeleteObject(hRgn);

break;

}

else {

//SetWindowText(hWnd, L"Didn't hit!");

}

DeleteObject(hRgn);

}

}

}

void DrawItemSupp(HDC hMemDC, HWND hWnd, LPDRAWITEMSTRUCT pdis,

short int\* globSelectedPointer, short int\* globCreatedPointer,

HBITMAP positive, HBITMAP negative) {

{

switch (pdis->itemAction) {

case ODA\_SELECT:

if ((pdis->itemState & ODS\_SELECTED) && !(\*globCreatedPointer)) {

(\*globSelectedPointer) = (\*globSelectedPointer) ? 0 : 1;

}

}

if ((\*globSelectedPointer)) {

SelectObject(hMemDC, negative);

}

else {

if (!(\*globCreatedPointer)) {

SelectObject(hMemDC, positive);

DeselectButton(hWnd, globCreatedPointer);

}

else

SelectObject(hMemDC, negative);

}

}

}

int getNotCreatedMedIndex(void) {

for (int i = 0; i < MAX\_MED; i++)

if (!alreadyCreated[i])

return i;

return MAX\_MED;

}

void DrawItemDependingOnParameters(HWND hWnd, HDC hMemDC, LPDRAWITEMSTRUCT pdis, WPARAM wParam) {

switch (wParam) {

case SRCBUTTONID:

DrawItemSupp(hMemDC, hWnd, pdis, &SRC\_IS\_SELECTED, &SRC\_CREATED, HSRCBitmap, HSRCMonoBitmap);

break;

case SNKBUTTONID:

DrawItemSupp(hMemDC, hWnd, pdis, &SNK\_IS\_SELECTED, &SNK\_CREATED, HSNKBitmap, HSNKMonoBitmap);

break;

case MEDBUTTONID: {

int res = getNotCreatedMedIndex();

if (res == MAX\_MED) {

DrawItemSupp(hMemDC, hWnd, pdis, &MED\_IS\_SELECTED, &MED\_CREATED, HBMPs\_Mono[res - 1], HBMPs\_Mono[res - 1]);

MED\_CREATED = 1;

}

else {

DrawItemSupp(hMemDC, hWnd, pdis, &MED\_IS\_SELECTED, &MED\_CREATED, HBMPs[res], HBMPs\_Mono[res]);

}

break;

}

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

BOOL CALLBACK RequestCapacityDlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDOK\_CAP: {

BOOL check;

int res = GetDlgItemInt(hDlg, IDC\_EDIT\_CAP, &check, FALSE);

if (check) {

if (res < 1 || res > 99) {

MessageBoxW(hDlg, L"Введите натуральное число от 1 до 99!", L"Некорректный ввод!", (UINT)NULL);

CAP\_RETURN\_RES = -3;

}

else {

CAP\_RETURN\_RES = res;

EndDialog(hDlg, 0);

}

}

else {

CAP\_RETURN\_RES = -1;

MessageBoxW(hDlg, L"Введите натуральное число от 1 до 99!", L"Некорректный ввод!", (UINT)NULL);

}

return TRUE;

}

case IDCANCEL\_CAP: {

EndDialog(hDlg, 0);

CAP\_RETURN\_RES = -2;

return TRUE;

}

}

break;

}

return FALSE;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

HWND hTextMaxFlow;

BOOL CALLBACK ShowResDlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_INITDIALOG:

hTextMaxFlow = GetDlgItem(hDlg, IDC\_STATIC\_MAX\_FLOW);

char str[256];

\_itoa(MAX\_FLOW\_RES, str, 10);

SetWindowTextA(hTextMaxFlow, str);

return TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDCANCEL:

case IDOK:

EndDialog(hDlg, 0);

return TRUE;

}

break;

case WM\_PAINT:

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hDlg, &ps);

DrawEverythingBackToRes(hdc);

EndPaint(hDlg, &ps);

return TRUE;

break;

}

return FALSE;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

HICON hIcon;

#define NEW\_FILE\_ID 699

#define OPEN\_FILE\_ID 700

#define SAVE\_FILE\_ID 701

#define SAVE\_FILE\_AS\_ID 702

#define EXIT\_ID 703

#define ABOUT\_ID 704

LRESULT CALLBACK MyWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_CREATE: {

SetClassLong(hWnd, GCL\_HICON, (LONG)hIcon);

HMENU hMenuBar = CreateMenu();

HMENU hFile = CreateMenu();

AppendMenu(hMenuBar, MF\_POPUP, (UINT)hFile, L"Файл");

AppendMenu(hFile, MF\_STRING, (UINT\_PTR)NEW\_FILE\_ID, L"Новый");

AppendMenu(hFile, MF\_STRING, (UINT\_PTR)OPEN\_FILE\_ID, L"Открыть");

AppendMenu(hFile, MF\_STRING, (UINT\_PTR)SAVE\_FILE\_ID, L"Сохранить");

AppendMenu(hFile, MF\_STRING, (UINT\_PTR)SAVE\_FILE\_AS\_ID, L"Сохранить как...");

AppendMenu(hFile, MF\_STRING, (UINT\_PTR)EXIT\_ID, L"Выход");

AppendMenu(hMenuBar, MF\_POPUP, (UINT\_PTR)ABOUT\_ID, L"Справка");

SetMenu(hWnd, hMenuBar);

HSRCBitmap = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(SRC\_BITMAP));

HSRCMonoBitmap = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(SRC\_MONO\_BITMAP));

HSelectedSRCBitmap = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(SELECTED\_SRC\_BITMAP));

HSNKBitmap = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(SNK\_BITMAP));

HSNKMonoBitmap = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(SNK\_MONO\_BITMAP));

HSelectedSNKBitmap = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(SELECTED\_SNK\_BITMAP));

HBMPs[0] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP1));

HBMPs[1] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP2));

HBMPs[2] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP3));

HBMPs[3] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP4));

HBMPs[4] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP5));

HBMPs[5] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP6));

HBMPs[6] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP7));

HBMPs[7] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP8));

HBMPs[8] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP9));

HBMPs[9] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP10));

HBMPs[10] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP11));

HBMPs[11] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP12));

HBMPs[12] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP13));

HBMPs[13] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP14));

HBMPs[14] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP15));

HBMPs[15] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP16));

HBMPs[16] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP17));

HBMPs[17] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP18));

HBMPs\_Selected[0] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP1\_S));

HBMPs\_Selected[1] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP2\_S));

HBMPs\_Selected[2] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP3\_S));

HBMPs\_Selected[3] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP4\_S));

HBMPs\_Selected[4] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP5\_S));

HBMPs\_Selected[5] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP6\_S));

HBMPs\_Selected[6] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP7\_S));

HBMPs\_Selected[7] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP8\_S));

HBMPs\_Selected[8] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP9\_S));

HBMPs\_Selected[9] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP10\_S));

HBMPs\_Selected[10] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP11\_S));

HBMPs\_Selected[11] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP12\_S));

HBMPs\_Selected[12] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP13\_S));

HBMPs\_Selected[13] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP14\_S));

HBMPs\_Selected[14] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP15\_S));

HBMPs\_Selected[15] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP16\_S));

HBMPs\_Selected[16] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP17\_S));

HBMPs\_Selected[17] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP18\_S));

HBMPs\_Mono[0] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP1\_M));

HBMPs\_Mono[1] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP2\_M));

HBMPs\_Mono[2] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP3\_M));

HBMPs\_Mono[3] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP4\_M));

HBMPs\_Mono[4] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP5\_M));

HBMPs\_Mono[5] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP6\_M));

HBMPs\_Mono[6] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP7\_M));

HBMPs\_Mono[7] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP8\_M));

HBMPs\_Mono[8] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP9\_M));

HBMPs\_Mono[9] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP10\_M));

HBMPs\_Mono[10] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP11\_M));

HBMPs\_Mono[11] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP12\_M));

HBMPs\_Mono[12] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP13\_M));

HBMPs\_Mono[13] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP14\_M));

HBMPs\_Mono[14] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP15\_M));

HBMPs\_Mono[15] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP16\_M));

HBMPs\_Mono[16] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP17\_M));

HBMPs\_Mono[17] = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(BMP18\_M));

for (int i = 0; i < MAX\_MED; i++)

alreadyCreated[i] = 0;

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_COMMAND: {

WORD word = LOWORD(wParam);

switch (word) {

case NEW\_FILE\_ID: {

ResetEverything();

InvalidateRect(hWnd, NULL, 0);

break;

}

case OPEN\_FILE\_ID: {

//wcscpy(currFileName, L"Souce.nk");

//SetWindowText(groupButton, currFileName);

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = currFileName;

ofn.lpstrFile[0] = '\0';

ofn.nMaxFile = sizeof(currFileName);

ofn.lpstrFilter = L"Сети\0\*.nw\0";

ofn.nFilterIndex = 0;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST | OFN\_HIDEREADONLY;

if (GetOpenFileName(&ofn) == TRUE) {

int res = readAllDataFromFile();

if (res == -1) {

MessageBox(hWnd, L"Произошла ошибка чтения!", L"Ошибка!", MB\_ICONERROR);

InvalidateRect(hWnd, NULL, 0);

}

else {

InvalidateRect(hWnd, NULL, 0);

}

SetWindowText(groupButton, currFileName);

}

else {

MessageBox(hWnd, L"Файл не был открыт!", L"Внимание!", MB\_ICONEXCLAMATION);

}

break;

}

case SAVE\_FILE\_ID: {

int res = SaveAllDataToFile();

if (res == -1) {

MessageBox(hWnd, L"Файл не был сохранен!", L"Внимание!", MB\_ICONERROR);

}

else {

MessageBox(hWnd, L"Запись прошла успешно!", L"Успех!", NULL);

}

break;

}

case SAVE\_FILE\_AS\_ID: {

ZeroMemory(&sfn, sizeof(sfn));

sfn.lStructSize = sizeof(sfn);

sfn.hwndOwner = hWnd;

sfn.lpstrFile = saveFileName;

sfn.lpstrFile[0] = '\0';

sfn.nMaxFile = sizeof(saveFileName);

sfn.lpstrFilter = L"Сети\0\*.nw\0";

sfn.nFilterIndex = 0;

sfn.lpstrFileTitle = NULL;

sfn.nMaxFileTitle = 0;

sfn.lpstrInitialDir = NULL;

sfn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST | OFN\_HIDEREADONLY;

if (GetSaveFileName(&sfn) == TRUE) {

if (wcsstr(saveFileName, L".nw") == NULL) wcscat(saveFileName, L".nw");

int res = writeAllDataToFile();

if (res == -1) {

MessageBox(hWnd, L"Ошибка доступа к файлу!", L"Внимание!", MB\_ICONERROR);

}

else {

MessageBox(hWnd, L"Запись прошла успешно!", L"Успех!", NULL);

}

}

else {

MessageBox(hWnd, L"Файл не был сохранен!", L"Внимание!", MB\_ICONEXCLAMATION);

}

break;

}

case EXIT\_ID: {

PostQuitMessage(0);

break;

}

case ABOUT\_ID: {

MessageBoxW(hWnd, L" Курсовая работа\n студента группы 951005\n Доведькоа Дианаислава Вадимовича\n Проверила Болтак Светлана Дианаимировна", L"Справка", MB\_ICONINFORMATION);

break;

}

case SRCBUTTONID:

if (SNK\_IS\_SELECTED || MED\_IS\_SELECTED) {

SRC\_IS\_SELECTED = 0;

return 0;

}

if (!SRC\_CREATED && SRC\_IS\_SELECTED) {

HCURSOR hCurs = LoadCursor(NULL, IDC\_CROSS);

SetClassLongPtr(hWnd, GCLP\_HCURSOR, (LONG)(hCurs));

}

else {

SRC\_IS\_SELECTED = 0;

}

break;

case SNKBUTTONID:

if (SRC\_IS\_SELECTED || MED\_IS\_SELECTED) {

SNK\_IS\_SELECTED = 0;

return 0;

}

if (!SNK\_CREATED && SNK\_IS\_SELECTED) {

HCURSOR hCurs = LoadCursor(NULL, IDC\_CROSS);

SetClassLongPtr(hWnd, GCLP\_HCURSOR, (LONG)(hCurs));

}

else {

SNK\_IS\_SELECTED = 0;

}

break;

case MEDBUTTONID: {

if (SRC\_IS\_SELECTED || SNK\_IS\_SELECTED) {

MED\_IS\_SELECTED = 0;

return 0;

}

if (!MED\_CREATED && MED\_IS\_SELECTED) {

HCURSOR hCurs = LoadCursor(NULL, IDC\_CROSS);

SetClassLongPtr(hWnd, GCLP\_HCURSOR, (LONG)(hCurs));

}

else {

MED\_IS\_SELECTED = 0;

}

break;

}

case MAXFLOWBUTTONID: {

//printCurrentStateDebugDifferent();

if (sourceP == NULL) {

MessageBoxW(hWnd, L"В введенном графе отсутствует исток!", L"Некорректный ввод!", (UINT)NULL);

}

else if (sinkP == NULL) {

MessageBoxW(hWnd, L"В введенном графе отсутствует сток!", L"Некорректный ввод!", (UINT)NULL);

}

else {

int N = getGreatestVertexNumber() + 3;

int\*\* w = (int \*\*)malloc(sizeof(int\*) \* e\_ind);

for (int i = 0; i < e\_ind; i++) {

w[i] = (int\*)malloc(sizeof(int) \* 3);

int u = edges[i].u->medNumber;

int v = edges[i].v->medNumber;

int c = edges[i].cap;

u += 2;

v = (v == -2) ? N : v + 2;

w[i][0] = u;

w[i][1] = v;

w[i][2] = c;

}

MAX\_FLOW\_RES = EdmondKarpAlgorithmFunc(N, e\_ind, w, &globalPointerToNetwork);

switch (MAX\_FLOW\_RES) {

case -1:

MessageBoxW(hWnd, L"В введеном графе из истока не исходит ни одной дуги!", L"Некорректный ввод!", (UINT)NULL);

break;

case -2:

MessageBoxW(hWnd, L"В введеном графе в сток не входит ни одна дуга!", L"Некорректный ввод!", (UINT)NULL);

break;

case -3:

MessageBoxW(hWnd, L"В введеном графе одна из промежуточных вершин \nне лежит на пути из истока в сток!", L"Некорректный ввод!", (UINT)NULL);

break;

default: {

DialogBox(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD\_RESDIALOGWIN), hWnd, ShowResDlgProc);

}

}

for (int i = 0; i < e\_ind; i++)

free(w[i]);

free(w);

}

}

}

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_LBUTTONDOWN: {

int x = GET\_X\_LPARAM(lParam);

int y = GET\_Y\_LPARAM(lParam);

HDC hdc = GetDC(hWnd);

RECT rect;

SetRect(&rect, DRAW\_AREA\_UPPER\_LEFT\_X + 30, DRAW\_AREA\_UPPER\_LEFT\_Y + 30, DRAW\_AREA\_LOWER\_RIGHT\_X - 30, DRAW\_AREA\_LOWER\_RIGHT\_Y - 30);

HRGN hRgn = CreateRectRgnIndirect(&rect);

SelectClipRgn(hdc, hRgn);

if (PtInRegion(hRgn, x, y)) {

SetFocus(hWnd);

SelectClipRgn(hdc, NULL);

ProcessClickInDrawArea(hWnd, hdc, x, y);

}

else {

DeselectButtonsIfAny(hWnd);

}

ReleaseDC(hWnd, hdc);

DeleteObject(hRgn);

//printCurrentStateDebug();

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_LBUTTONUP: {

DRAG\_ACTIVE = 0;

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_PAINT: {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

//DrawAllVerticesBack(hdc);

DrawEverythingBack(hdc);

RECT rect;

SetRect(&rect, 600, 15, 750, 100);

DrawText(hdc, L"Поиск \rмаксимального потока\r в сети\rалгоритмом\rЭдмондса-Карпа", -1, &rect, DT\_CENTER | DT\_WORDBREAK);

SetRect(&rect, 600, 125, 750, 150);

DrawText(hdc, L"Исток сети", -1, &rect, DT\_CENTER | DT\_SINGLELINE);

SetRect(&rect, 600, 230, 750, 270);

DrawText(hdc, L"Промежуточная вершина", -1, &rect, DT\_CENTER | DT\_WORDBREAK);

SetRect(&rect, 600, 355, 750, 375);

DrawText(hdc, L"Сток сети", -1, &rect, DT\_CENTER | DT\_SINGLELINE);

EndPaint(hWnd, &ps);

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_MOUSEMOVE: {

int x = GET\_X\_LPARAM(lParam);

int y = GET\_Y\_LPARAM(lParam);

HRGN hRgn;

RECT drawAreaRect;

SetRect(&drawAreaRect, DRAW\_AREA\_UPPER\_LEFT\_X + 30, DRAW\_AREA\_UPPER\_LEFT\_Y + 30, DRAW\_AREA\_LOWER\_RIGHT\_X - 30, DRAW\_AREA\_LOWER\_RIGHT\_Y - 30);

hRgn = CreateRectRgnIndirect(&drawAreaRect);

if (PtInRegion(hRgn, x, y)) {

if (DRAG\_ACTIVE && wParam == MK\_LBUTTON && firstChosenP != NULL) {

SetRect(&(firstChosenP->vRect), x - 21, y - 21, x + 21, y + 21);

HDC hdc = GetDC(hWnd);

DrawEverythingBack(hdc);

//DrawAllVerticesBack(hdc);

ReleaseDC(hWnd, hdc);

}

}

DeleteObject(hRgn);

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case WM\_DRAWITEM: {

LPDRAWITEMSTRUCT pdis;

pdis = (LPDRAWITEMSTRUCT)lParam;

HDC hMemDC = CreateCompatibleDC(pdis->hDC);

switch (wParam) {

case SRCBUTTONID:

if (SNK\_IS\_SELECTED || MED\_IS\_SELECTED) {

SRC\_IS\_SELECTED = 0;

return TRUE;

}

break;

case SNKBUTTONID:

if (SRC\_IS\_SELECTED || MED\_IS\_SELECTED) {

SNK\_IS\_SELECTED = 0;

return TRUE;

}

break;

case MEDBUTTONID:

if (SRC\_IS\_SELECTED || SNK\_IS\_SELECTED) {

MED\_IS\_SELECTED = 0;

return TRUE;

}

}

DrawItemDependingOnParameters(hWnd, hMemDC, pdis, wParam);

FillRect(pdis->hDC, &pdis->rcItem, (HBRUSH)(COLOR\_WINDOW + 1));

TransparentBlt(pdis->hDC, 0, 0, 41, 41,

hMemDC, 0, 0, 41, 41, COLOR\_TO\_BE\_TRANSPARENT);

DeleteDC(hMemDC);

return TRUE;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define BACKSPACE\_VIRTUAL\_CODE 0x08

case WM\_KEYDOWN: {

if (wParam == BACKSPACE\_VIRTUAL\_CODE) {

if (firstChosenP != NULL) {

HDC hdc = GetDC(hWnd);

DeleteVertexFromArray(hdc, firstChosenP);

firstChosenP = NULL;

DeselectButtonsIfAny(hWnd);

ReleaseDC(hWnd, hdc);

}

}

//printCurrentStateDebug();

return 0;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

}

return DefWindowProc(hWnd, uMsg, wParam, lParam);

}

INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR lpCmdLine, INT nCmdShow)

{

WNDCLASSEX wcex; //Оконный класс для окна нашего приложения

HWND hWnd; //Главное окно нашего приложения

MSG msg; //Структура для получения и отправки сообщений

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

//////////////////////////////////////////////

////////////////////////////////////////////// //Заполнение полей оконного класса

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = 0;

wcex.lpfnWndProc = MyWindowProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = 0;

wcex.hCursor = LoadCursor(0, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = L"MyWindowClass";

hIcon = (HICON) LoadImage(hInstance, MAKEINTRESOURCE(IDB\_ICON), IMAGE\_ICON, 16, 16, LR\_DEFAULTCOLOR);

wcex.hIconSm = NULL;

//////////////////////////////////////////////

//////////////////////////////////////////////

RegisterClassEx(&wcex); //Регистрация оконного класса

//Создание главного окна приложения

hWnd = CreateWindowEx(0, L"MyWindowClass", L"Программное средство для нахождения максимального потока в сети",

WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_MINIMIZEBOX,

0, 0, 800, 600, 0, 0, hInstance, NULL);

//Создание рамки для области рисования

groupButton = CreateWindowEx(WS\_EX\_WINDOWEDGE, L"Button", currFileName,

WS\_VISIBLE | WS\_CHILD | WS\_BORDER | BS\_GROUPBOX,

DRAW\_AREA\_UPPER\_LEFT\_X - 15,

DRAW\_AREA\_UPPER\_LEFT\_Y - 15,

DRAW\_AREA\_WIDTH + 25,

DRAW\_AREA\_HEIGHT + 25, hWnd, NULL, NULL, NULL);

//Создание рамки для меню

HWND groupButtonAn = CreateWindowEx(WS\_EX\_WINDOWEDGE, L"Button", NULL,

WS\_VISIBLE | WS\_CHILD | WS\_BORDER | BS\_GROUPBOX,

MENU\_AREA\_UPPER\_LEFT\_X,

MENU\_AREA\_UPPER\_LEFT\_Y,

MENU\_AREA\_WIDTH,

MENU\_AREA\_HEIGHT, hWnd, NULL, NULL, NULL);

//Создание кнопки с источником сети

HWND srcButton = CreateWindowEx(WS\_EX\_WINDOWEDGE, L"Button", L"Btn1",

WS\_VISIBLE | WS\_CHILD | BS\_OWNERDRAW,

655,

150,

42,

42, hWnd, (HMENU)SRCBUTTONID, NULL, NULL);

//Создание кнопки с промежуточными вершинами

HWND midButton = CreateWindowEx(WS\_EX\_WINDOWEDGE, L"Button", L"Btn2",

WS\_VISIBLE | WS\_CHILD | BS\_OWNERDRAW,

655,

270,

42,

42, hWnd, (HMENU)MEDBUTTONID, NULL, NULL);

//Создание кнопки со стоком сети

HWND snkButton = CreateWindowEx(WS\_EX\_WINDOWEDGE, L"Button", L"Btn3",

WS\_VISIBLE | WS\_CHILD | BS\_OWNERDRAW,

655,

375,

42,

42, hWnd, (HMENU)SNKBUTTONID, NULL, NULL);

//Создание кнопки для расчета максимального потока

HWND maxFlowButton = CreateWindowEx(WS\_EX\_WINDOWEDGE, L"Button", L"Найти максимальный поток сети",

WS\_VISIBLE | WS\_CHILD | BS\_MULTILINE,

605,

440,

140,

70,

hWnd, (HMENU)MAXFLOWBUTTONID, NULL, NULL);

//Показ главного окна приложения

ShowWindow(hWnd, nCmdShow);

//Цикл обработки сообщений

while (GetMessage(&msg, 0, 0, 0))

{

DispatchMessage(&msg);

}

GdiplusShutdown(gdiplusToken);

//Возвращение результата работы

return msg.wParam;

}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 016 ПЗ | | | | Пояснительная записка | | | | 89 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951005 016 ПД | | | | Схема программы | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 016 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программное средство  для нахождения максимального потока сети.  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Доведько В.В. |  |  | Т |  | |  | 89 | 89 |
| Пров. | | Болтак С.В. |  |  | Кафедра ПОИТ  гр. 951005 | | | | | |
|  | |  |  |  |