МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Систем и Технологий

Кафедра «Информационные системы»

Дисциплина «Основы алгоритмизации и программирования»

**КУРСОВАЯ РАБОТА(ПРОЕКТ)**

Тема Визуализация алгоритма поиска минимального остового дерева в графе

\_\_\_\_\_\_

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Дозорова А. А/

подпись инициалы, фамилия

Курс второй Группа ПИбд-12

Направление 09.03.04 «Программная инженерия»

Руководитель к.т.н., доцент Гуськов Г.Ю.

должность, ученая степень, ученое звание, фамилия, имя, отчество

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Оценка:

Ульяновск

2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий

Кафедра «Информационные системы»

Дисциплина «Основы алгоритмизации и программирования»

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ(ПРОЕКТ)**

Студенту ПИбд-21 Дозоровой А. А.

группа фамилия, инициалы

Тема проекта (работы)\_Визуализация алгоритма поиска минимального остового дерева в графе

Срок сдачи законченного проекта «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Исходные данные к проекту УлГТУ, курс лекций по дисциплине «Основы алгоритмизации и программирования», интернет-источники

Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Введение.

Планирование разработки проекта. Техническое задание. Планирование и формирование списка задач.

Описание проектирования. Техническое проектирование. Представление диаграмм с их описанием.

Описание процедур процесса конструирования.

Алгоритмическое обеспечение, описание алгоритма. Описание созданных проектов, классов, интерфейсов и т.п.

Описание технологий, использованных при разработке.

Пример работы приложения.

Качество конструирования. План и итоги тестирования.

Заключение

Скриншоты разработанного программного продукта. Примеры работы пирложения.

Руководитель к.т.н, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ / Гуськов Г.Ю. /

должность подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **/** **\_\_\_\_\_\_\_\_\_/**

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ  
руководителя на курсовую работу (проект)**

студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Дозоровой Алены Алексеевны\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

фамилия, имя и отчество

Факультет информационных систем и технологий группа ПИбд-12 курс второй

Дисциплина «Основы алгоритмизации и программирования»

Тема проекта Визуализация алгоритма поиска минимального остового дерева в графе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель к.т.н. доцент кафедры «Информационные системы» /Гуськов Г. Ю. /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г.

**Оглавление**

[Введение 3](#_Toc102141070)

[Планироваине разработки проекта 4](#_Toc102141071)

[Техническое задание 4](#_Toc102141072)

[Планирование и формирование списка задач 4](#_Toc102141073)

[Описание проектирования 6](#_Toc102141074)

[Техническое проектирование 6](#_Toc102141075)

[Описание процедур процесса конструирования 7](#_Toc102141076)

[Алгоритмическое обеспечение 7](#_Toc102141077)

[Описание алгоритмической реализации 7](#_Toc102141078)

[Описание технологий, использованных при разработке 8](#_Toc102141079)

[Пример работы приложения 10](#_Toc102141080)

[Тестирование 14](#_Toc102141081)

[Заключение 16](#_Toc102141082)

[Список литературы 18](#_Toc102141083)

[Приложение 19](#_Toc102141084)

[Исходный код программы: 19](#_Toc102141085)

# Введение

Темой данной курсового проекта визуализация алгоритма поиска минимального остовного дерева в графе.

Существует несколько алгоритмов построения остовного дерева. Данный проект позволит наглядно продемонстрировать работу алгоритмов Прима, Краскала, Борувки и обратного удаления для лучшего понимания данной темы. Кроме визуализации, программа позволит самому создать неориентированный взвешенный граф, сгенерировать его и изменить.

Разработанная программа станет хорошим пособием для изучения алгоритмов построения минимального остовного дерева.

# Планироваине разработки проекта

## Техническое задание

Разработать приложение, позволяющее визуализировать алгоритмы построения минимального остовного дерева. Реализовать сами алгоритмы, а так же построение графа случайным образом, по заданным ребрам, изменение количества вершин, ребер.

Применить при разработке известные алгоритмы построения минимального остовного дерева. Реализовать конкретные классы, а именно: класс, представляющий граф, классы, предсавляющие вершины и ребра; класс(ы) графического интерфейса приложения. Грамотно спроектировать архитектуру приложения.

Выполнить эмпирическое (опытное доказательство соответствующего роста времени) обоснование сложности. Выполнить тестирование приложения в использовании, корректность работы, производительность.

* Изучение различных реализаций визуализации алгоритма;
* Разработка начальной архитектуры приложения, анализ возможностей для расширения;
* Разработка базовой реализации алгоритма;
* Анализ её достоинств и недостатков, поиск путей решения возникших проблем;
* Реализовать графический интерфейс приложения;
* Предоставить пользователю возможность настройки графа

## Планирование и формирование списка задач

В таблице 1 представлен полный список задач с оценкой трудоемкости и сроками реализации, сформированный для обеспечения контроля и успешного заверешения проекта.

Таблица 1. Исходный план задач с оценкой трудоемкости и сроками реализации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Этап разработки и задачи | Планируемые временные затраты, T0 | Фактические временные затраты, T1 | Поставленные подзадачи, W0 | Решеные подзадачи, W1 | Предполагаемые сроки реализации |
| 1. Реализация графа и алгоритмов поиска минимального остовного дерева | | | | | 01.03.22 |
| * 1. Описание концепции и основной идеи алгоритма | 1-2 часа |  |  |  | 10.2.22 |
| * 1. Анализ существующих решений, подбор подходящей композиции алгоритмов | 6-11 часов |  |  |  | 17.2.22 |
| * 1. Теоретическая оценка возможности реализации алгоритма | 6-8 часа |  |  |  | 17.2.22 |
| * 1. Подготовка к реализации алгоритма | 3-5 часов |  |  |  | 20.2.22 |
| 1. Верификация алгоритма | | | | | 01.03.23 |
| * 1. Декомпозиция алгоритма | 0.5-1 час |  |  |  | 21.02.22 |
| * 1. Реализация алгоритма | 7-12 часов |  |  |  | 22.02.22 |
| * 1. Теоретическое доказательство корректности | 6-10 часов |  |  |  | 23.02.22 |
| * 1. Подготовка тестовых вариантов графа | 1-2 часа |  |  |  | 24.02.22 |
| * 1. Тестирование алгоритма | 1-2 часа |  |  |  | 24.02.22 |
| * 1. Исправление возможных ошибок | 1-2 часа |  |  |  | 24.02.22 |
| 1. Разработка структуры и архитектуры ПО | | | | | 25.02.22 |
| * 1. Разделение ПО на модули | 0.5-1 час |  |  |  | 25.02.22 |
| 1. Создание UI | 5-15 часов |  |  |  | 15.03.22 |
| 1. Итоговое тестирование | 1.5-2.5 часа |  |  |  | 15.03.22 |
| Итог | 39,5-70,5 |  |  |  |  |

Минимальная оценка трудозатрат 39,5 часов, что довольно много, для выполнения одним человеком в качестве курсового проекта. Наверняка, фактическая оценка трудозатрат будет сильно ближе к верхней теоретической оценке, или, может быть даже больше, т.к. не учтены все подзадачи, которые могут значительно увеличить требуемое на разработку время. С учетом того, что у исполнителя есть опыт в разработке алгоритмов и графического интерфейса, проект можно считать выполнимым, с учетом наличия нагрузки со стороны других предметов, в противном случае, полное выполнение работы, вероятно, не состоялось.

# Описание проектирования

## Техническое проектирование

Программа имеет следующую структуру:

1. Класс интерфейса FormMain — отвечает за графический интерфейс приложения.

Методы:

* public void DesDelete() реализует алгоритм обратного удаления
* public void Boruvka() реализует алгоритм Борувки
* public void Kruskal() реализует алгоритм Краскала
* Bitmap Draw(List<Edge>edges) визуализирует граф
* void Prim(int begin) реализует алгоритм Прима
* private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e) переключает визуализацию между алгоритмов
* private void button\_Random\_Click(object sender, EventArgs e) реализует генерацию графа
* private void buttonCreate\_Click(object sender, EventArgs e) генерирует поля для ввода ребер пользователем
* private void buttonSave\_Click(object sender, EventArgs e) формирует граф по введенным пользователем ребрам
* private void button\_Add\_Click(object sender, EventArgs e) реализует добавление вершин
* private void buttonDelete\_Click(object sender, EventArgs e) реализует удаление вершины
* private void buttonAddEdge\_Click(object sender, EventArgs e) реализует добавление ребра
* private void buttonDeleteEdge\_Click(object sender, EventArgs e) реализует удаление ребра
* public void Text\_Graph() реализует текстовое описание графа

1. Класс Graph — класс служащий для создания графа, где хранятся списки вершин и ребер.

Методы:

* public void Create(int count, bool? ed) создает визуализацию графа, его генерацию
* public void AddVertex(Vertex vertex) добавляет вершину
* public void AddEdge(Vertex vertexFirst, Vertex vertexSecond, int w) добавляет ребро
* public void Update() – обновление графа, объявление всех вершин непройденными
* public bool Svyznost(List<Edge> edge) проверяет граф на связность
* public void DFD(Vertex i, List<Edge> e) рекурсивно проходит граф в глубину
* public List<Edge> Sort(List<Edge> edges, bool des) сортирует список ребер по их весу
* public int GetWeight(List<Edge> edges) считает вес графа
* public String GetVertex(List<Edge> edges) возвращает вершины, которые соединяет конкретное ребро

1. Edge – класс, реализующий ребро графа
2. Vertex – класс, реализующий вершину
3. Program – точка запуска программы

# Описание процедур процесса конструирования

Инструменты:

Язык программирования – C#, .NET framework.

Разработка структуры и архитектуры ПО проходит непосредственно после реализации графа и алгоритмов построения, так как это является основополагающим.

# Алгоритмическое обеспечение

Алгоритмы, используемые для визуализации таковы: основой является алгоритм построения, который возвращает список ребер и функция рисования воспроизводит построение на экране.

## Описание алгоритмической реализации

Рассмотрим последовательно все реализованные алгоритмы построения минимального остовного дерева:

Алгоритм Прима

На вход алгоритма подаётся связный неориентированный граф, где для каждого ребра задана его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Алгоритм Краскала

На вход алгоритма подается связный неориентированный граф, где для каждого ребра задан вес. Формируется пустое множество ребер.

Пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

Алгоритм Борувки

Работа алгоритма состоит из нескольких итераций, каждая из которых состоит в последовательном добавлении рёбер к [остовному лесу](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BB%D0%B5%D1%81" \o "Остовный лес) графа, до тех пор, пока лес не превратится в [дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)), то есть, лес, состоящий из одной компоненты связности.

То есть, от каждой вершины проводится ребро с минимальным весом до тех пор, пока граф не станет связным

Алгоритм обратного удаления

На вход подается неориентированный связный граф. Пока граф связный, из него удаляются ребра с максимальным весом.

**Сложность алгоритмов**

Для алгоритмов генерации графа, его визуализации, получения текстового описания сложность равна O(n), так как все реализовано проходом по листу в цикле.

Для алгоритмов построения сложность следующая:

Для алгоритма обратного удаления O(n^2) – так как происходит обход по двум листам

Для алгоритма Борувки O(n)

Для алгоритма Краскала аналогично O(n)

Для алгоритма Прима O(n^2) из-за вложенного цикла

Полученные сложности практически совпадают со сложностями классических реализаций данных алгоритмов.

# Описание технологий, использованных при разработке

При разработке данного программного продукта были задействованы различные инструменты, с помощью которых были реализованы все изначально заявленные возможности программы.

Используемый язык программирования – C#.

Работа с графическими объектами(изображениями) с помощью WinForms.

Преимущества:

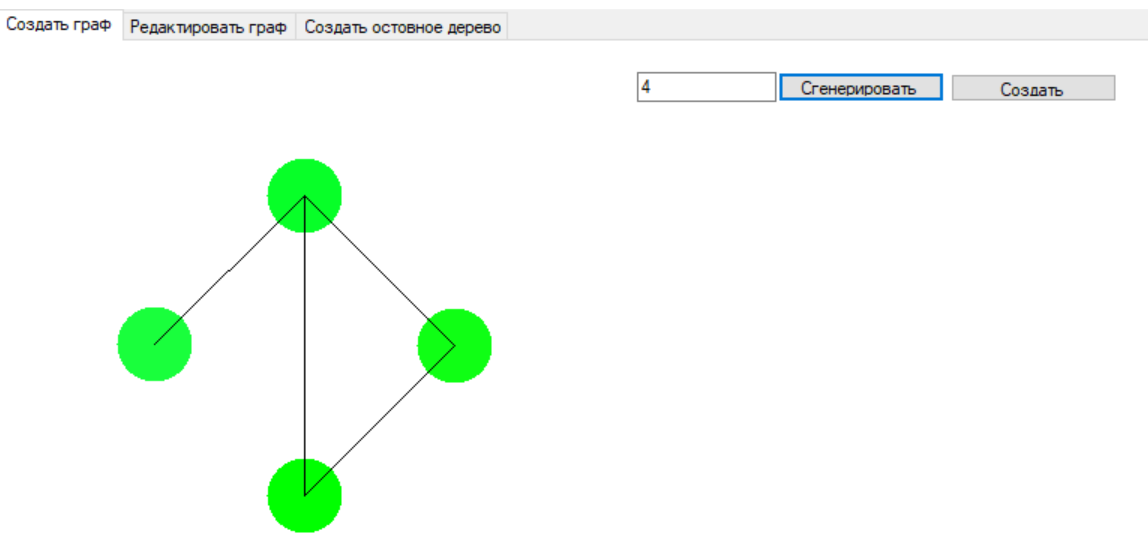
* представление изображения как битную матрицу (Bitmap), что максимально удобно при изменении изображений

Недостатки:

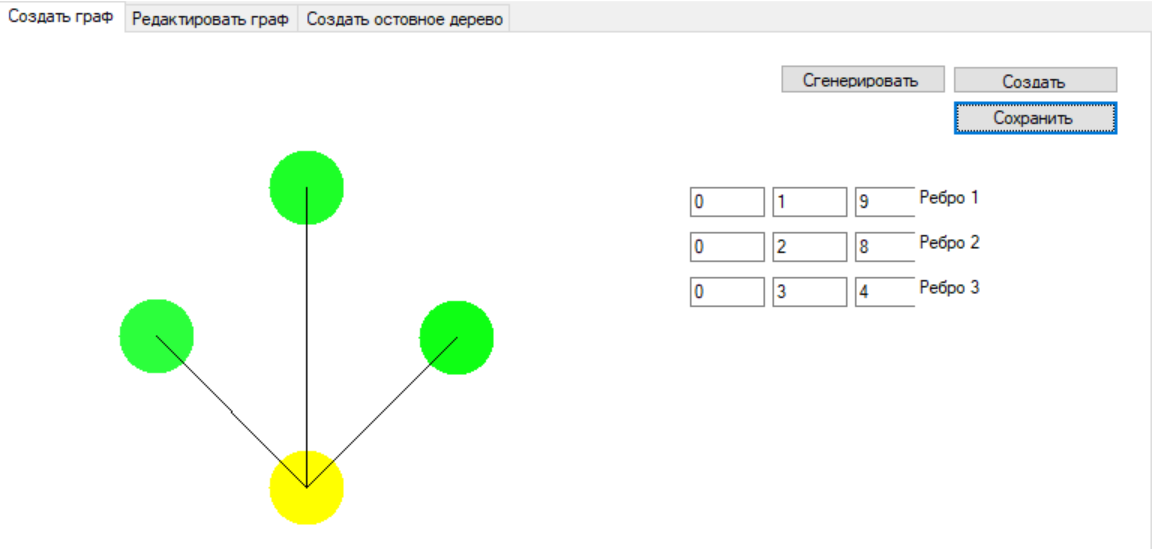
* сложность работы с подписями на изображении.

# Пример работы приложения

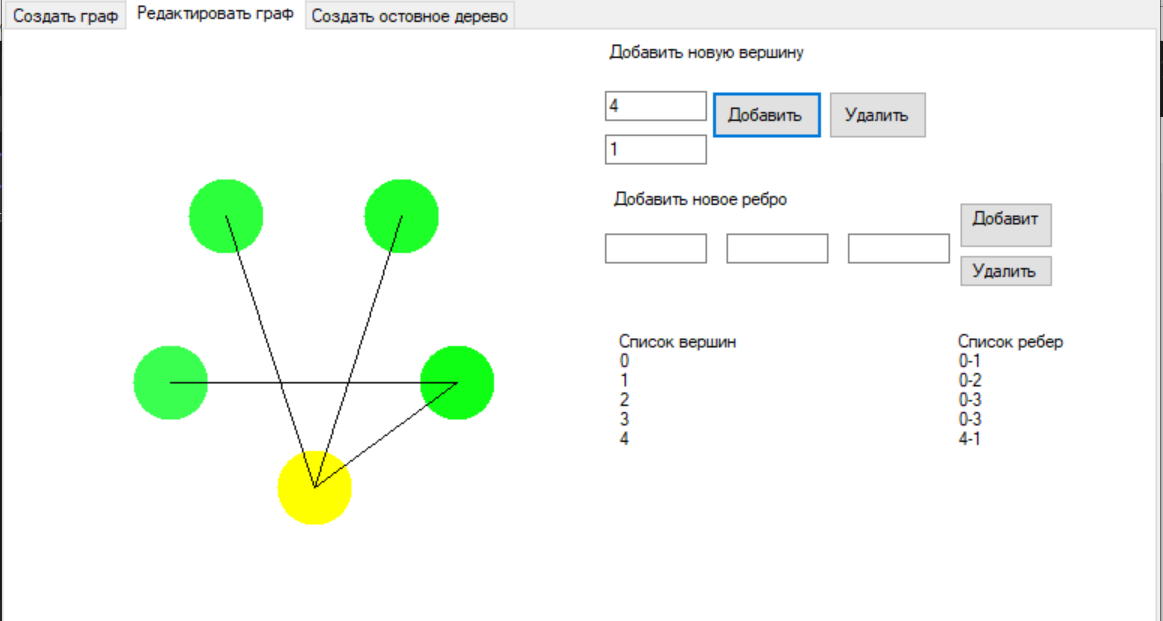
Генерация графа:



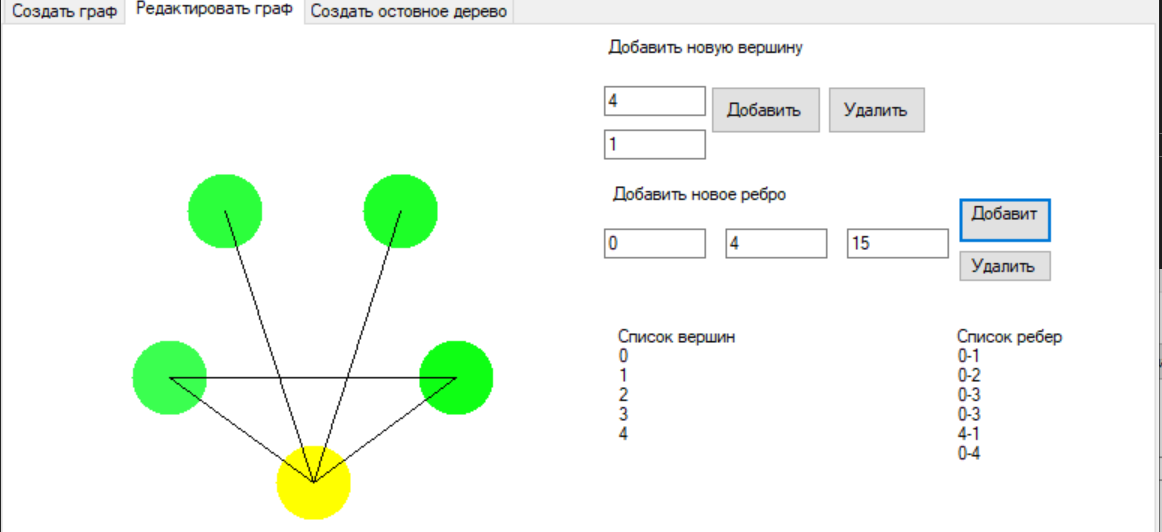
Создание графа



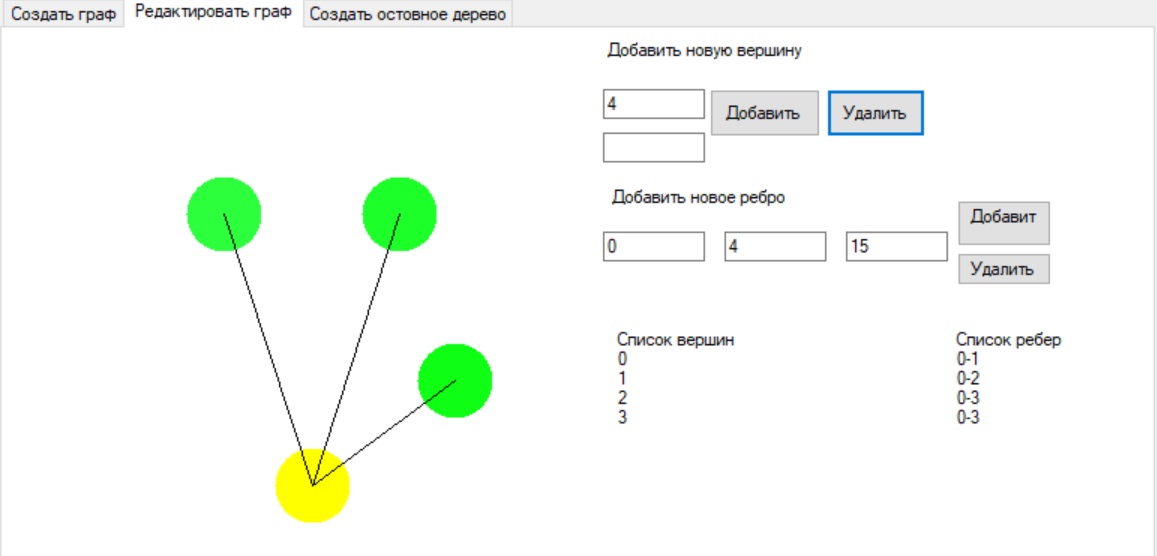
Добавление новой вершины:



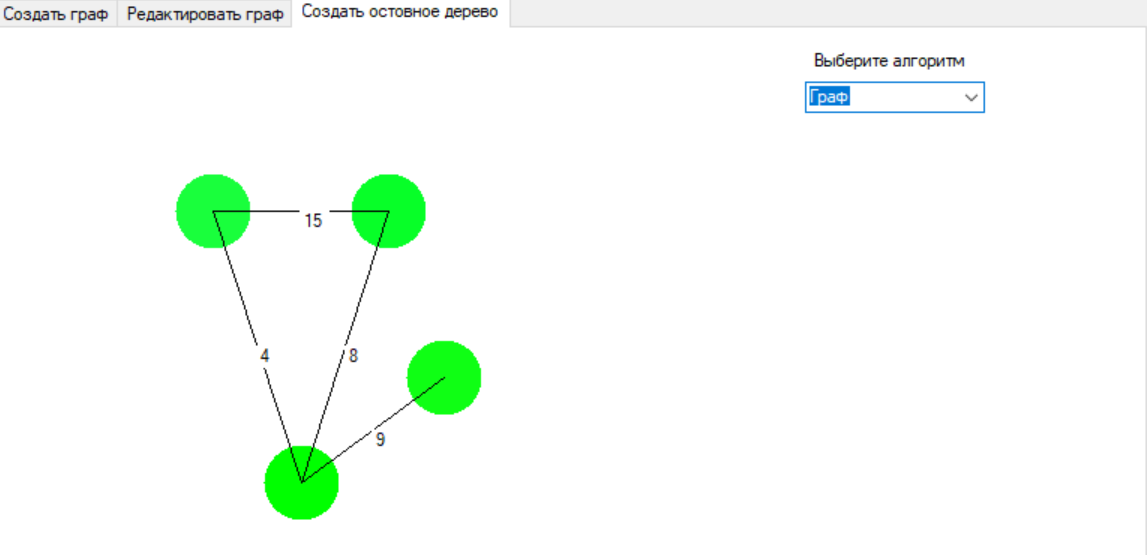
Добавление ребра



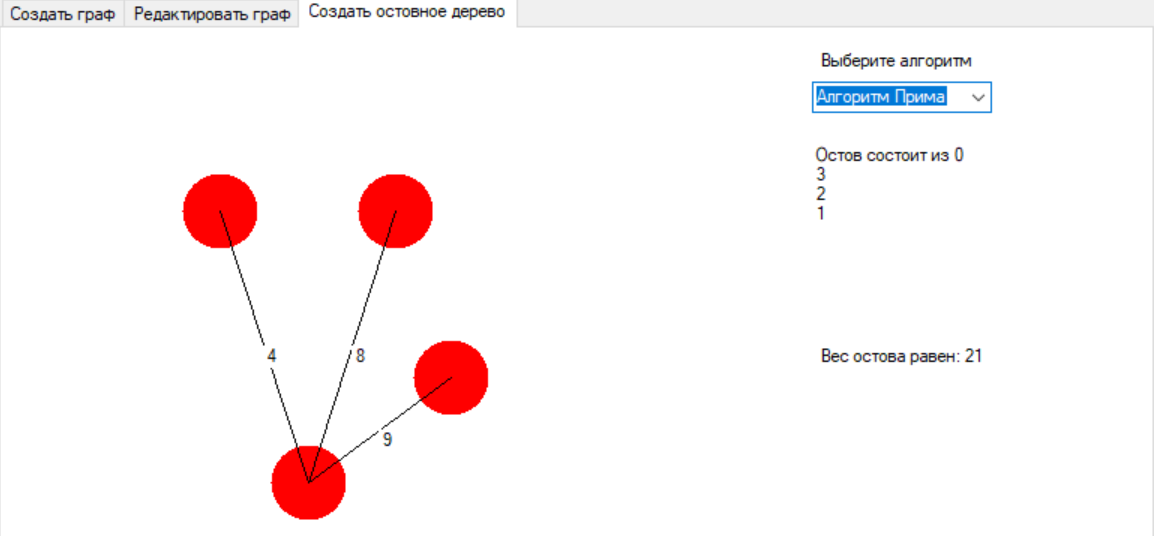
Удаление ребра

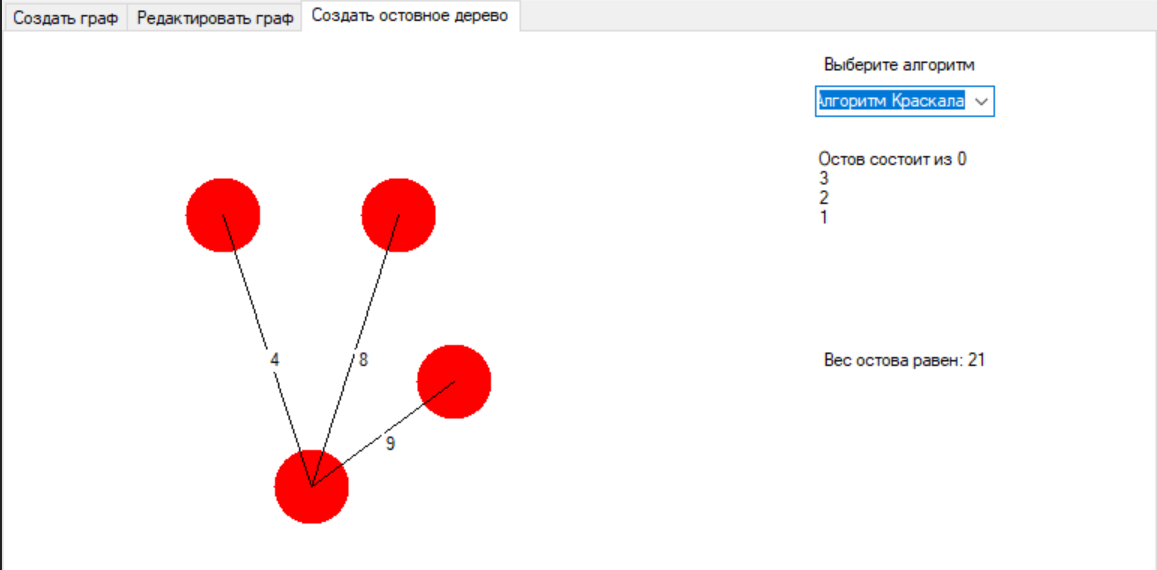


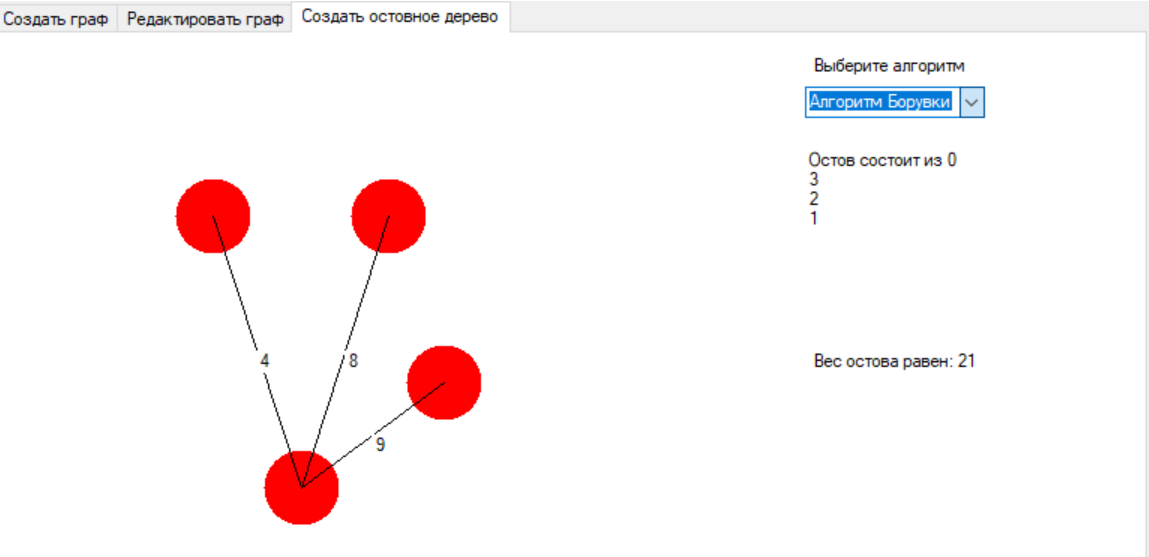
Сам граф с весами ребер

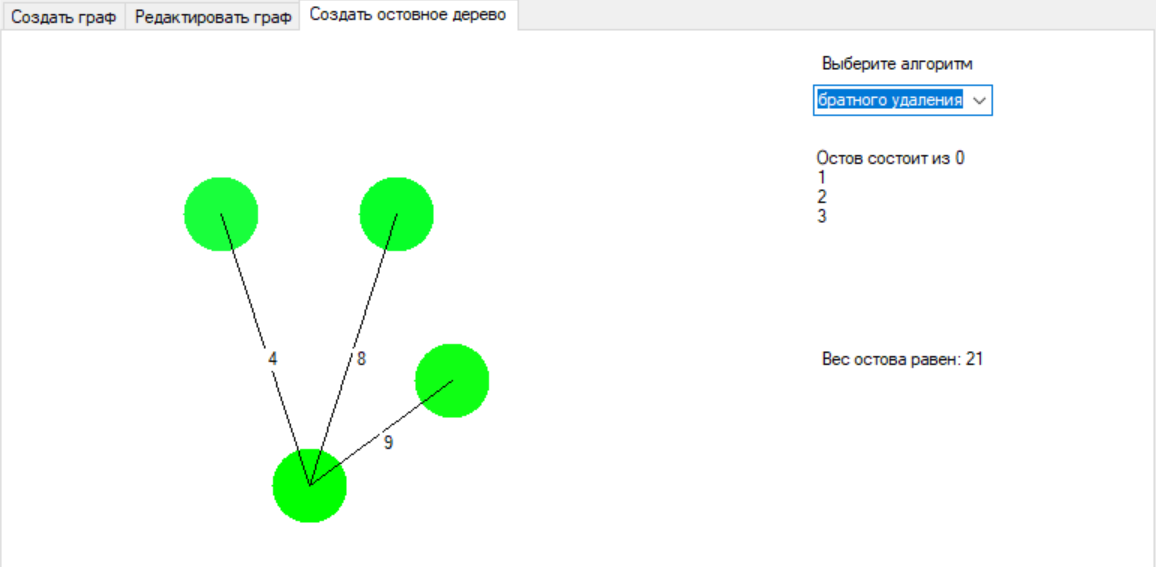


Алгоритмы построения





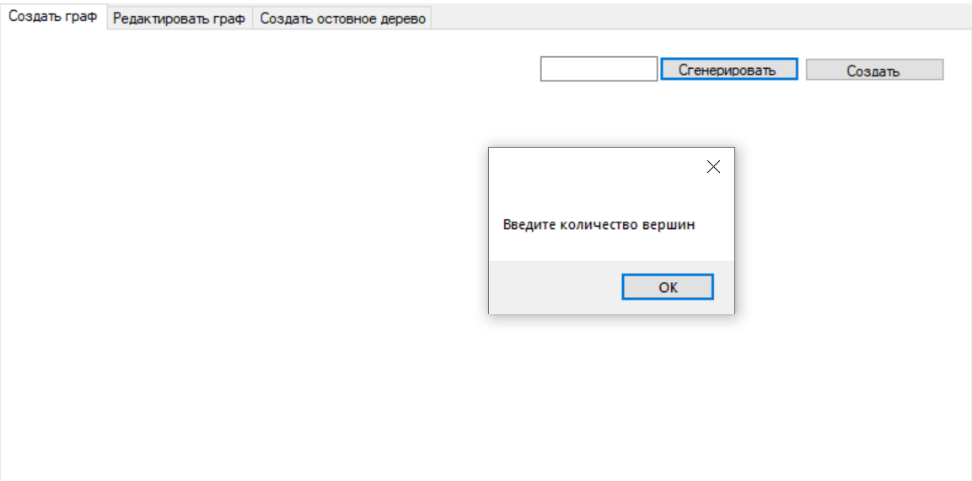




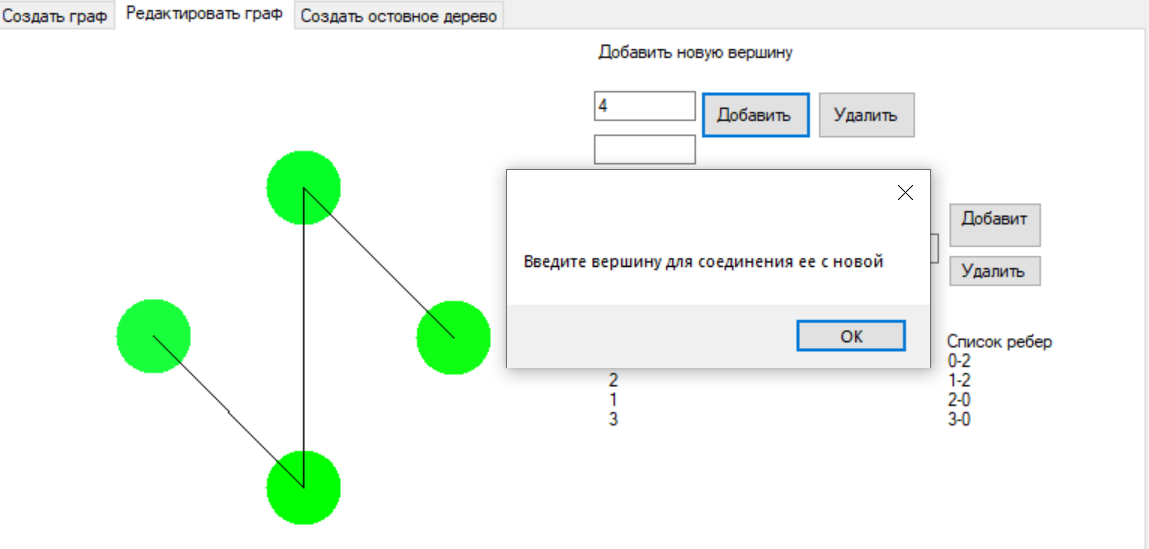
## Тестирование

В программе реализованы проверки на неверный ввод пользователем данных.

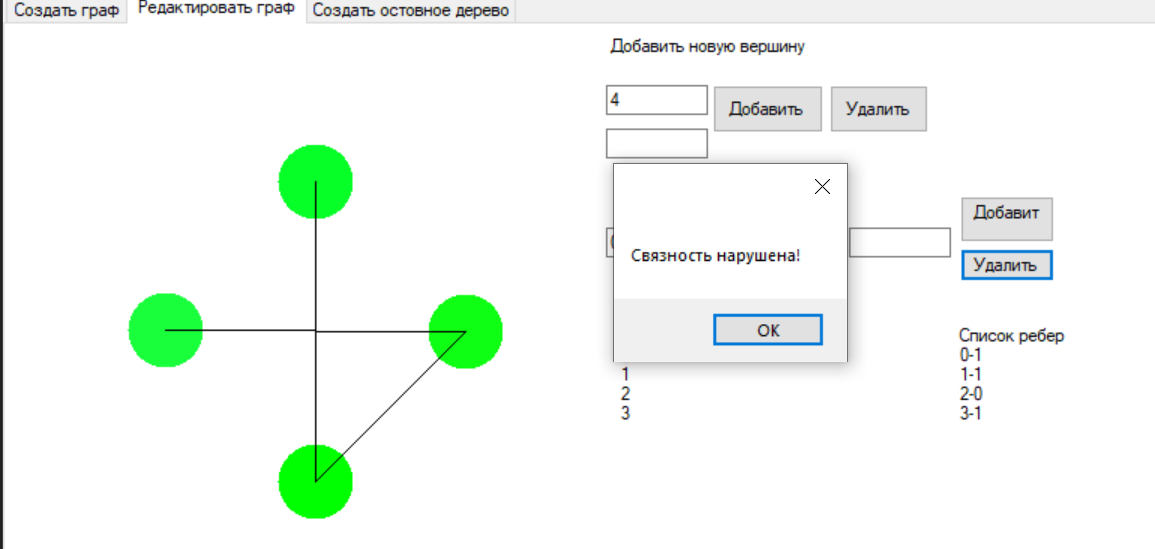
При отсутвии количества вершин:



При добавлении вершины отсутвие вершины связанной с ней, чтобы граф оставался связным:



Удаление ребра, нарушающего связность:



В случае недостатка данных при добавлении ребра, вес ребра вводится по умолчанию.

# Заключение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Этап разработки и задачи | Планируемые временные затраты, T0 | Фактические временные затраты, T1 | Поставленные подзадачи, W0 | Решеные подзадачи, W1 |
| 1. Реализация графа и алгоритмов поиска минимального остовного дерева | | | | |
| * 1. Описание концепции и основной идеи алгоритма | 1-2 часа | 1 | 1 | 1 |
| * 1. Анализ существующих решений, подбор подходящей композиции алгоритмов | 6-11 часов | 6 | 5 | 5 |
| * 1. Теоретическая оценка возможности реализации алгоритма | 6-8 часа | 6 | 2 | 2 |
| * 1. Подготовка к реализации алгоритма | 3-5 часов | 3 | 2 | 2 |
| 1. Верификация алгоритма | | | | |
| * 1. Декомпозиция алгоритма | 0.5-1 час | 1 | 4 | 4 |
| * 1. Реализация алгоритма | 7-12 часов | 7 | 5 | 5 |
| * 1. Теоретическое доказательство корректности | 6-10 часов | 5 | 4 | 4 |
| * 1. Подготовка тестовых вариантов графа | 1-2 часа | 1 | 5 | 5 |
| * 1. Тестирование алгоритма | 1-2 часа | 1 | 3 | 3 |
| * 1. Исправление возможных ошибок | 1-2 часа | 1 | 1 | 1 |
| 1. Разработка структуры и архитектуры ПО | | | | |
| * 1. Разделение ПО на модули | 0.5-1 час | 0.5 | 1 | 1 |
| 1. Создание UI | 5-15 часов | 7 | 3 | 3 |
| 1. Итоговое тестирование | 1.5-2.5 часа | 1 | 1 | 1 |
| Итог | 39,5-70,5 | 40,5 |  |  |

Некоторые реализационные работы были проведены быстрее нижней границы планируемых затрат. Но в результате разработка уложилась в предсказанный период времени.

Проект завершен в поставленные сроки, с приближением к нижней границе заявленного времени на реализацию проекта, сроки не нарушены.

Проведена демонстрация проекта с описанием реализованной функциональности.

В процессе написания курсового проекта были:

* улучшены навыки владением ООП;
* улучшено знание языка C# и платформы .NET в целом;
* улучшены навыки алгоритмического мышления;
* получены новые знания в области работы с графами;
* получены навыки по работе с графикой в C#;
* освоены принципы проектирования программ;
* получены навыки работы с программами контроля версий (Git)

Разработанное приложение уже является законченным демонстрационным материалом, что позволяет использовать его на практике. При желании, приложение можно дополить другими алгоритмами поиска минимального остовного дерева, что не изменяет факта полноты разработанного приложения.

# Список литературы

1. Использование элемента управления WinForms [Электронный ресурс] // Microsoft. – Режим доступа: https://docs.microsoft.com/ru-ru/sql/reporting-services/application-integration/using-the-winforms.
2. Асимптотический анализ алгоритмов [Электронный ресурс] // Habrahabr. - 2006-2018 г.. Режим доступа: <https://habr.com/ru/post/78728/>
3. Минимальное остовное дерево [Электронный ресурс] <https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>
4. Материалы лекций курса «Основы алгоритмизации и программирования»

# Приложение

## Исходный код программы:

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Ostovnoe\_derevo

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new FormMain());

}

}

}

FormMain.cs

using System;

using System.Threading;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Ostovnoe\_derevo

{

public partial class FormMain : Form

{

public FormMain()

{

InitializeComponent();

}

Graph graph = new Graph();

List<Edge> PrimMST = new List<Edge>();

List<Edge> KruskalMST = new List<Edge>();

List<Edge> BoruvkaMST = new List<Edge>();

List<Edge> DeleteMST = new List<Edge>();

private void Form1\_Load(object sender, EventArgs e)

{

Text\_Graph();

pictureBox3.Image = pictureBox2.Image;

}

public void DesDelete()

{

DeleteMST = new List<Edge>();

List<Edge> edges =new List<Edge>();

foreach (Edge e in graph.edges)

{

DeleteMST.Add(e);

edges.Add(e);

}

edges = graph.Sort(edges, false);

foreach (Edge e in edges)

{

List<Edge> temp = new List<Edge>();

foreach(Edge e2 in DeleteMST)

{

temp.Add(e2);

}

temp.Remove(e);

if(graph.Svyznost(temp) == true)

{

DeleteMST.Remove(e);

pictureBox1.Image = Draw(DeleteMST);

pictureBox1.Refresh();

Thread.Sleep(1000);

}

else

{

pictureBox1.Image = Draw(DeleteMST);

pictureBox1.Refresh();

Thread.Sleep(1000);

continue;

}

}

}

public int FindRoot(int node, int[]parent)

{

var root = node;

while(root!=parent[root])

{

root = parent[root];

}

while(node!=root)

{

var oldParent = parent[node];

parent[node] = root;

node = oldParent;

}

return root;

}

public void Boruvka()

{

BoruvkaMST = new List<Edge>();

List<Edge> List\_edges = new List<Edge>();

foreach(Vertex v in graph.vertices)

{

List\_edges = graph.edges.Where(e=>e.FirstPoint == v || e.SecondPoint == v).Select(e=>e).ToList();

int min = 1000;

Edge min\_e = null;

foreach(Edge e in List\_edges)

{

if(e.Weight<min)

{

min = e.Weight;

min\_e = e;

}

}

if(min\_e!=null)

{

min\_e.FirstPoint.color = Color.Red;

min\_e.SecondPoint.color = Color.Red;

BoruvkaMST.Add(min\_e);

pictureBox1.Image = Draw(BoruvkaMST);

pictureBox1.Refresh();

Thread.Sleep(1000);

}

}

}

public void Kruskal()

{

KruskalMST = new List<Edge>();

List<Edge> edgesNew = new List<Edge>();

foreach (Edge e in graph.edges)

{

edgesNew.Add(e);

}

edgesNew = graph.Sort(edgesNew, true);

var parent = Enumerable.Range(0, graph.vertices.Count).ToArray();

foreach(Edge e in edgesNew)

{

var startNodeRoot = FindRoot(e.FirstPoint.Name, parent);

var endNodeRoot = FindRoot(e.SecondPoint.Name, parent);

if(startNodeRoot != endNodeRoot)

{

e.FirstPoint.color = Color.Red;

e.SecondPoint.color = Color.Red;

KruskalMST.Add(e);

parent[endNodeRoot] = startNodeRoot;

pictureBox1.Image = Draw(KruskalMST);

pictureBox1.Refresh();

Thread.Sleep(1000);

}

}

}

Bitmap Draw(List<Edge>edges)

{

pictureBox1.Image = null;

Bitmap bitmap = new Bitmap(793, 432);

Graphics line = Graphics.FromImage(bitmap);

foreach(Label label in pictureBox1.Controls)

{

pictureBox1.Controls.Remove(label);

}

SolidBrush \_Srush = (SolidBrush)Brushes.Violet;

foreach (Vertex ver in graph.vertices)

{

ToolTip tip = new ToolTip();

\_Srush.Color = ver.color;

line.FillEllipse(\_Srush, ver.X - 25, ver.Y - 25, 50, 50);

tip.Tag = ver;

}

foreach (Edge edge in edges)

{

Pen pen = new Pen(Color.Black);

line.DrawLine(pen, new Point(edge.FirstPoint.X, edge.FirstPoint.Y), new Point(edge.SecondPoint.X, edge.SecondPoint.Y));

Label label = new Label();

label.BackColor = Color.White;

label.Size = new Size(20, 15);

label.Text = edge.Weight.ToString();

label.Location = new Point((edge.FirstPoint.X + edge.SecondPoint.X) / 2, (edge.FirstPoint.Y + edge.SecondPoint.Y) / 2);

pictureBox1.Controls.Add(label);

}

return bitmap;

}

void Prim(int begin)

{

PrimMST = new List<Edge>();

List<Edge> notUsedE = new List<Edge>(graph.edges.Count);

foreach (Edge e in graph.edges)

{

notUsedE.Add(e);

}

List<Vertex> usedV = new List<Vertex>();

List<Vertex> notUsedV = new List<Vertex>();

for (int i = 0; i < graph.vertices.Count; i++)

notUsedV.Add(graph.vertices[i]);

Random rand = new Random();

usedV.Add(graph.vertices[begin]);

notUsedV.Remove(graph.vertices[begin]);

while (notUsedV.Count > 0)

{

int minE = -1;

for (int i = 0; i < notUsedE.Count; i++)

{

if ((usedV.IndexOf(notUsedE[i].FirstPoint) != -1) && (notUsedV.IndexOf(notUsedE[i].SecondPoint) != -1) ||

(usedV.IndexOf(notUsedE[i].SecondPoint) != -1) && (notUsedV.IndexOf(notUsedE[i].FirstPoint) != -1))

{

if (minE != -1)

{

if (notUsedE[i].Weight < notUsedE[minE].Weight)

minE = i;

}

else

minE = i;

}

}

if (usedV.IndexOf(notUsedE[minE].FirstPoint) != -1)

{

usedV.Add(notUsedE[minE].SecondPoint);

notUsedV.Remove(notUsedE[minE].SecondPoint);

}

else

{

usedV.Add(notUsedE[minE].FirstPoint);

notUsedV.Remove(notUsedE[minE].FirstPoint);

}

notUsedE[minE].FirstPoint.color = Color.Red;

notUsedE[minE].SecondPoint.color = Color.Red;

PrimMST.Add(notUsedE[minE]);

pictureBox1.Image = Draw(PrimMST);

pictureBox1.Refresh();

Thread.Sleep(1000);

notUsedE.RemoveAt(minE);

}

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

pictureBox1.Image = null;

switch (comboBox1.Text)

{

case "Граф":

graph.Update();

pictureBox1.Image = Draw(graph.edges);

pictureBox1.Refresh();

labelMST.Text = null;

labelWeight.Text = null;

break;

case "Алгоритм Прима":

graph.Update();

pictureBox1.Refresh();

Prim(0);

labelMST.Text = null;

labelMST.Text = "Остов состоит из " + graph.GetVertex(PrimMST);

labelWeight.Text = "Вес остова равен: "+graph.GetWeight(PrimMST).ToString();

break;

case "Алгоритм Краскала":

graph.Update();

pictureBox1.Refresh();

Kruskal();

labelMST.Text = "Остов состоит из " + graph.GetVertex(KruskalMST);

labelWeight.Text = "Вес остова равен: " + graph.GetWeight(KruskalMST).ToString();

break;

case "Алгоритм Борувки":

graph.Update();

pictureBox1.Refresh();

Boruvka();

labelMST.Text = "Остов состоит из " + graph.GetVertex(PrimMST);

labelWeight.Text = "Вес остова равен: " + graph.GetWeight(BoruvkaMST).ToString();

break;

case "Алгоритм обратного удаления":

graph.Update();

pictureBox1.Refresh();

DesDelete();

labelMST.Text = "Остов состоит из " + graph.GetVertex(DeleteMST);

labelWeight.Text = "Вес остова равен: " + graph.GetWeight(DeleteMST).ToString();

break;

}

}

private void button\_Random\_Click(object sender, EventArgs e)

{

graph.edges.Clear();

graph.vertices.Clear();

if(textBox\_Count.Text != "")

{

graph.Create(Convert.ToInt32(textBox\_Count.Text), null);

pictureBox2.Image = Draw(graph.edges);

pictureBox3.Image = Draw(graph.edges);

}

else

{

MessageBox.Show("Введите количество вершин");

}

}

private void buttonCreate\_Click(object sender, EventArgs e)

{

graph.edges.Clear();

graph.vertices.Clear();

buttonSave.Visible = true;

var controls = panel2.Controls.OfType<TextBox>().Select(c => c).ToList();

foreach(TextBox text in controls)

{

panel2.Controls.Remove(text);

}

if (textBox\_Count.Text!="")

{

int count = Convert.ToInt32(textBox\_Count.Text)-1;

graph.Create(count, null);

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

{

TextBox ver1 = new TextBox();

ver1.Name = "ver1" + i;

ver1.Location = new Point(50, 100 + i \* 30);

ver1.Size = new Size(50, 10);

TextBox ver2 = new TextBox();

ver2.Name = "ver2" + i;

ver2.Location = new Point(105, 100 + i \* 30);

ver2.Size = new Size(50, 10);

TextBox verd = new TextBox();

verd.Name = "verd" + i;

verd.Location = new Point(160, 100 + i \* 30);

verd.Size = new Size(50, 10);

Label label = new Label();

label.Text = "Ребро " + (i + 1);

label.Location = new Point(200, 100 + i \* 30);

panel2.Controls.Add(label);

panel2.Controls.Add(ver1);

panel2.Controls.Add(ver2);

panel2.Controls.Add(verd);

}

}

else

{

MessageBox.Show("Введите количество вершин");

}

}

private void buttonSave\_Click(object sender, EventArgs e)

{

graph.edges.Clear();

int count = Convert.ToInt32(textBox\_Count.Text);

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

{

graph.AddVertex(new Vertex(i, false, 0, 0, Color.AliceBlue));

}

var controls = panel2.Controls.OfType<TextBox>().Select(c=>c).ToList();

int f = count;

int s = count;

int w = 0;

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

{

foreach(TextBox text in controls)

{

if(text.Name== "ver1" + i)

{

f = Convert.ToInt32(text.Text);

}

if (text.Name == "ver2" + i)

{

s = Convert.ToInt32(text.Text);

}

if (text.Name == "verd" + i)

{

w = Convert.ToInt32(text.Text);

}

}

try

{

graph.AddEdge(graph.vertices[f], graph.vertices[s], w);

}

catch

{

MessageBox.Show("Вводите значения от 0 до числа вершин - 1");

return;

}

}

graph.Create(count, true);

pictureBox2.Image = Draw(graph.edges);

pictureBox3.Image = Draw(graph.edges);

}

private void button\_Add\_Click(object sender, EventArgs e)

{

Random random = new Random();

SolidBrush \_Srush = (SolidBrush)Brushes.Violet;

graph.vertices.Add(new Vertex(graph.vertices.Count, false, 0, 0, \_Srush.Color));

if (textBox\_edge\_New\_Ver.Text!="")

{

try

{

graph.AddEdge(graph.vertices[graph.vertices.Count-1], graph.vertices[Convert.ToInt32(textBox\_edge\_New\_Ver.Text)], random.Next(100));

}

catch

{

MessageBox.Show("Нет такой вершины");

}

}

else

{

MessageBox.Show("Введите вершину для соединения ее с новой");

}

graph.Create(graph.vertices.Count, true);

pictureBox3.Image = Draw(graph.edges);

Text\_Graph();

}

private void buttonDelete\_Click(object sender, EventArgs e)

{

List<Edge> edges = new List<Edge>();

int name = Convert.ToInt32(textBox\_Add\_Ver.Text);

foreach(Edge edge in graph.edges)

{

if(edge.FirstPoint.Name != name && edge.SecondPoint.Name != name)

{

edges.Add(edge);

}

}

graph.edges = edges;

try

{

graph.vertices.Remove(graph.vertices[name]);

pictureBox3.Image = Draw(graph.edges);

Text\_Graph();

}

catch

{

MessageBox.Show("Нет такой вершины");

}

}

private void buttonAddEdge\_Click(object sender, EventArgs e)

{

int f = Convert.ToInt32(textBox\_f.Text);

int s = Convert.ToInt32(textBox\_s.Text);

int w;

if(textBox\_w.Text=="")

{

w = 1;

}

else w = Convert.ToInt32(textBox\_w.Text);

graph.AddEdge(graph.vertices[f], graph.vertices[s], w);

pictureBox3.Image = Draw(graph.edges);

Text\_Graph();

}

private void buttonDeleteEdge\_Click(object sender, EventArgs e)

{

int f = Convert.ToInt32(textBox\_f.Text);

int s = Convert.ToInt32(textBox\_s.Text);

var d = graph.edges.Where(c => c.FirstPoint == graph.vertices[f] && c.SecondPoint == graph.vertices[s]).Select(c => c).ToList();

List<Edge> edges = new List<Edge>();

graph.edges.ForEach(c => edges.Add(c));

edges.Remove(d[0]);

if(!graph.Svyznost(edges))

{

MessageBox.Show("Связность нарушена!");

return;

}

graph.edges = edges;

pictureBox3.Image = Draw(graph.edges);

Text\_Graph();

}

public void Text\_Graph()

{

if(pictureBox2.Image!=null)

{

labelVer.Text = "Список вершин" + Environment.NewLine + graph.GetVertex(graph.edges);

labelEdge.Text = "Список ребер" + Environment.NewLine;

foreach (Edge e in graph.edges)

{

labelEdge.Text += e.FirstPoint.Name.ToString() + "-" + e.SecondPoint.Name.ToString() + Environment.NewLine;

}

}

}

private void tabControl1\_Selected(object sender, TabControlEventArgs e)

{

Text\_Graph();

}

}

}

Graph.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

using System.Threading.Tasks;

namespace Ostovnoe\_derevo

{

public class Graph

{

public List<Edge> edges = new List<Edge>();

public List<Vertex> vertices = new List<Vertex>();

public void Create(int count, bool? ed)

{

vertices.Clear();

Random r = new Random();

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

{

SolidBrush \_Srush = (SolidBrush)Brushes.Violet;

\_Srush.Color = Color.FromArgb(255%(Math.Abs(255 - i \* 15)+1), 255, 255%(Math.Abs(255 - 1 - i \* 20)+1));

double x = 200;

double y = 200;

double ax = Math.Sin((2\*Math.PI / (count))\*i)\*100;

double ay = Math.Cos((2 \* Math.PI / (count))\*i)\*100;

AddVertex(new Vertex(i, false, (int)(x+ax), (int)(y+ay), \_Srush.Color));

}

if(ed != true)

{

while (!Svyznost(edges))

{

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

{

int ind = r.Next(0, count - 1);

while (ind == i)

{

ind = r.Next(0, count - 1);

}

foreach (Edge e in edges)

{

if (e.FirstPoint == vertices[ind] && e.SecondPoint == vertices[i])

{

ind = r.Next(0, count - 1);

}

}

AddEdge(vertices[i], vertices[ind], r.Next(100));

}

}

}

else

{

List<Edge> e = new List<Edge>();

foreach (Edge \_e in edges)

{

Vertex v1 = vertices.Where(c => c.Name == \_e.FirstPoint.Name).Select(c => c).ToList()[0];

Vertex v2 = vertices.Where(c => c.Name == \_e.SecondPoint.Name).Select(c => c).ToList()[0];

e.Add(new Edge(v1, v2, \_e.Weight));

}

edges = e;

}

}

public void AddVertex(Vertex vertex)

{

vertices.Add(vertex);

}

public void AddEdge(Vertex vertexFirst, Vertex vertexSecond, int w)

{

edges.Add(new Edge(vertexFirst, vertexSecond, w));

}

public void Update()

{

Color color = new Color();

foreach (Vertex v in vertices)

{

v.color = Color.FromArgb(255%(255 - Math.Abs(255 - vertices.IndexOf(v) \* 15)+1), 255, 255%(Math.Abs(255 - 1 - vertices.IndexOf(v) \* 20)+1));

v.IsChecked = false;

}

}

public bool Svyznost(List<Edge> edge)

{

Update();

DFD(vertices[0], edge);

foreach(Vertex v in vertices)

{

if(v.IsChecked==false)

{

return false;

}

}

return true;

}

public void DFD(Vertex i, List<Edge> e)

{

vertices[vertices.IndexOf(i)].IsChecked = true;

foreach(Edge u in e)

{

if(u.FirstPoint == i && u.SecondPoint.IsChecked == false)

{

DFD(u.SecondPoint, e);

}

else if (u.SecondPoint == i && u.FirstPoint.IsChecked == false)

{

DFD(u.FirstPoint, e);

}

}

}

public List<Edge> Sort(List<Edge> edges, bool des)

{

if (des)

{

for (int i = 0; i < edges.Count; i++)

{

bool good = true;

for (int j = 0; j < edges.Count - 1; j++)

{

if (edges[j].Weight > edges[j + 1].Weight)

{

Edge temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

good = false;

}

}

if (good)

{

break;

}

}

}

else

{

for (int i = 0; i < edges.Count; i++)

{

bool good = true;

for (int j = 0; j < edges.Count - 1; j++)

{

if (edges[j].Weight < edges[j + 1].Weight)

{

Edge temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

good = false;

}

}

if (good)

{

break;

}

}

}

return edges;

}

public int GetWeight(List<Edge> edges)

{

int sum = 0;

foreach (Edge e in edges)

{

sum += e.Weight;

}

return sum;

}

public String GetVertex(List<Edge> edges)

{

String ver = "";

foreach (Edge e in edges)

{

if (!ver.Contains(e.FirstPoint.Name.ToString()))

{

ver+=e.FirstPoint.Name.ToString()+Environment.NewLine;

}

if (!ver.Contains(e.SecondPoint.Name.ToString()))

{

ver += e.SecondPoint.Name.ToString() + Environment.NewLine;

}

}

return ver;

}

}

}

Edge.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ostovnoe\_derevo

{

public class Edge

{

public Vertex FirstPoint { get; private set; }

public Vertex SecondPoint { get; private set; }

public int Weight { get; private set; }

public Edge(Vertex first, Vertex second, int valueOfWeight)

{

FirstPoint = first;

SecondPoint = second;

Weight = valueOfWeight;

}

public override string ToString()

{

return FirstPoint.Name+" "+SecondPoint.Name;

}

}

}

Vertex.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Ostovnoe\_derevo

{

public class Vertex

{

public int Name { get; private set; }

public bool IsChecked { get; set; }

public int X { get; set; }

public int Y { get; set; }

public Color color;

public Vertex(int name, bool check, int x, int y, Color c)

{

Name = name;

IsChecked = check;

X = x; Y = y;

color = c;

}

public override string ToString()

{

return Name.ToString();

}

int[] Points()

{

return new int[] { X, Y };

}

}

}