|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий (ИТ) | |
| Кафедра корпоративных информационных систем (КИС**)**  **КУРСОВАЯ РАБОТА**  по дисциплине  «Структуры и алгоритмы обработки данных»  **Тема курсовой работы**: Программное обеспечение для вычисления кратчайшего расстояния между двумя обозначенными вершинами в графе   |  |  | | --- | --- | | Студент группы ИКБО-06-17 | Янгуразов Рашид Алиевич | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) | |  |  | | Руководитель курсовой работы | Андрианова Елена Гельевна | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) | |  |  | | Работа представлена к защите | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. | |  |  | | Допущен к защите | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. | | |

|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий (ИТ) | |

Кафедра корпоративных информационных систем (КИС**)**

Утверждаю

Заведующий кафедрой КИС

\_\_\_\_\_\_\_\_\_\_\_\_\_А.Б. Петров

«21» декабря 2018 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

«Структуры и алгоритмы обработки данных»

|  |  |
| --- | --- |
| Студент Янгуразов Рашид Алиевич | Группа ИКБО-06-17 |

**Тема работы:** Программное обеспечение для вычисления кратчайшего расстояния между двумя обозначенными вершинами в графе

**Исходные данные:** среда разработки на языке Python, принципы объектно-ориентированного подхода в проектировании и программировании, алгоритмы поиска кратчайшего пути в графе

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

1. Изучение графовых структур данных, их разновидностей и операций с ними.
2. Проектирование структуры приложения, использующего для хранения всех видов данных массивы подходящих типов.
3. Проектирование UI для представления графа в графической нотации.
4. Реализация и тестирование средствами языка Python.
5. Написание расчетно-пояснительной записки.

|  |  |  |  |
| --- | --- | --- | --- |
| **Срок представления к защите курсовой работы:** | | до «20» декабря 2018 г. | |
|  | |  | |
| **Задание на курсовую работу выдал** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Андрианова Е.Г.) |
|  | | «07» сентября 2018 г. | |
| **Задание на курсовую работу получил** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Янгуразов Р.А.) |

|  |  |  |  |
| --- | --- | --- | --- |
| **№ этапа** | **Этапы выполнения**  **Курсовой Работы** | **Сроки выполнения** | **Отметка о выполнении** |
|  | Изучение графовых структур данных, их разновидностей и операций с ними | 15.10.2018 |  |
|  | Проектирование структуры приложения, использующего для хранения всех видов данных массивы подходящих типов, и UI для представления графа в графической нотации | 30.10.2018 |  |
|  | Реализация и тестирование программного приложения | 15.11.2018 |  |
|  | Отчет по курсовой работе в виде расчетно-пояснительной записки | 20.12.2018 |  |

**Содержание**

[Реферат 6](#_Toc533979183)

[Введение 7](#_Toc533979184)

[1. Теоретическая часть 8](#_Toc533979185)

[1.1 Граф как абстрактная математическая единица. 8](#_Toc533979186)

[1.2 Понятие кратчайшего расстояния 9](#_Toc533979187)

[1.3 Алгоритм Дейкстры 10](#_Toc533979188)

[1.4 Библиотека Tkinter или графика на Python 11](#_Toc533979189)

[2. Проектная часть 12](#_Toc533979190)

[2.1 Постановка задачи 12](#_Toc533979191)

[2.2 Структура программного приложения 13](#_Toc533979192)

[2.2.1 Модуль, включающий служебные классы 13](#_Toc533979193)

[2.2.2 Модуль, отвечающий за рисование в графическом окне 13](#_Toc533979194)

[2.2.3 Модуль, отвечающий за реализацию алгоритма Дейкстры и графа как программной единицы 14](#_Toc533979195)

[2.3 Проектирование графического интерфейса 14](#_Toc533979196)

[3. Реализация и тестирование программного приложения 15](#_Toc533979197)

[3.1 Реализация программного приложения 15](#_Toc533979198)

[3.1.1 Хранение деталей о графе в объекте класса Graph 15](#_Toc533979199)

[3.1.2 Реализация алгоритма Дейкстры 17](#_Toc533979200)

[3.1.3. Реализация проекта приложения 19](#_Toc533979201)

[3.1.4. Реализация графического интерфейса 19](#_Toc533979202)

[3.2. Тестирование ПО 20](#_Toc533979203)

[3.3. Тестирование исходного кода на соответствие PEP 8 22](#_Toc533979204)

[Заключение 23](#_Toc533979205)

[Список литературы 24](#_Toc533979206)

[Приложения 25](#_Toc533979207)

[Приложение №1. Демонстрация работы приложения 25](#_Toc533979208)

[Приложение №2. Листинги исходных текстов программы 26](#_Toc533979209)

[Приложение №3. Скриншоты с презентации к курсовой работе 33](#_Toc533979210)

# Реферат

Курсовая работа выполнена в объеме 36 страниц, и включает в себя 3 главных раздела: теоретическую, проектную и практическую части, а также графические материалы: 1 таблицу, 25 изображений.

Теоретическая часть посвящена графу как абстрактной математической единице, его практическому применению. Рассмотрено понятие кратчайшего расстояния и дана характеристика общеизвестному алгоритму, позволяющему его вычислять. Также эта часть включает в себя сведения о графической библиотеке Tkinter и перечисление модулей этой библиотеки, нашедших применение в программе.

В проектной части отражается постановка задачи, показывается разделение программы на структурные единицы, даётся информация о необходимых базовых составляющих программы – реализованных классах, методах и графическом интерфейсе.

Экспериментальная часть включает в себя этап тестирования программы, соответствие стандарту PEP 8. Результаты тестирований занесены в соответствующую таблицу. Для наглядной демонстрации корректности работы написанного программного обеспечения в приложение №1 курсовой работы будут приложены скриншоты, демонстрирующие работу программы.

# Введение

Данные, хранящиеся в компьютерной памяти, - это конечные последовательности нулей и единиц, которые принято называть битами. Биты объединяются в байты, слова. Каждый байт или слово имеет в оперативной памяти свой порядковый номер – адрес.

Все данные, используемые для решения практических задач, бывают нескольких типов, причем понятие тип связывается не только с представлением данных в адресном пространстве, но и способом их обработки.

Любые данные могут быть отнесены к одному из двух типов: простому и сложному. Форма представления первых зависит от архитектуры вычислительной машины, форма вторых же – от характера конкретных задач. К простым данным можно отнести, к примеру, числа целого или вещественного типов. Примером же сложной структуры данных является граф, вернее, его разного рода вариации.

Цель курсовой работы: разработка программного обеспечения для вычисления кратчайшего расстояния между двумя обозначенными вершинами в графе с помощью компьютерного алгоритма Эдсгера Вибе Дейкстры; приобретение и улучшение навыков в некоторых областях. К этим областям относятся обертка приложений в графический интерфейс с помощью библиотеки Tkinter, правильное использование структур данных на языке Python, работа с такими интегрированными средами разработки, как IDLE Python и JetBrains PyCharm Edu.

# Теоретическая часть

## Граф как абстрактная математическая единица.

Граф — это совокупность двух конечных множеств: множества точек (от английского — vertex), которые называются вершинами, и множества ребер (от английского — edge). Каждый элемент — это упорядоченная пара элементов множества , сами вершины и называются концевыми точками или концами ребра . Граф называется конечным, если множества и E конечны.

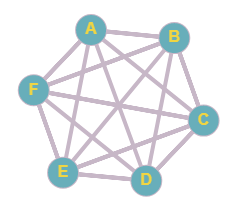


Рис 1. Пример неориентированного графа

Это определение графа должно быть дополнено в одном важном отношении. В определении ребра можно принимать или не принимать во внимание порядок расположения двух его концов. Если этот порядок несущественен, т. е. если , то говорят, что есть неориентированное ребро; если же этот порядок существенен, то называется ориентированным ребром (ориентированное ребро имеет другое название - дуга). В последнем случае называется также начальной вершиной, а – конечной вершиной ребра . Граф называется неориентированным, если каждое его ребро не ориентировано, и ориентированным, если ориентированы все его ребра. В ряде случаев

естественно рассматривать смешанные графы, имеющие как ориентированные, так и неориентированные ребра.

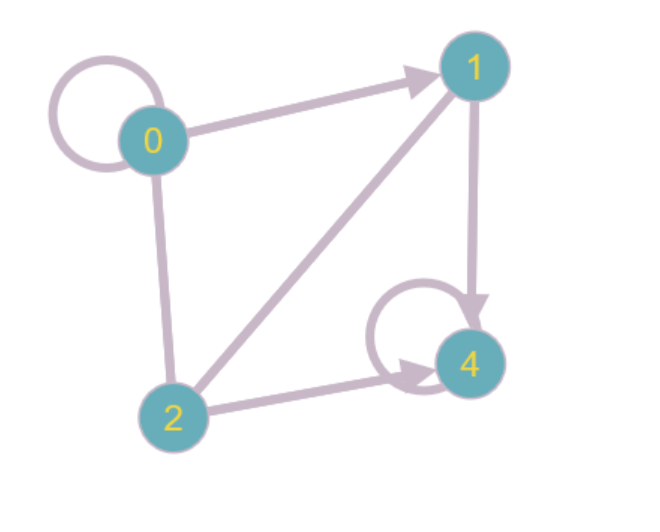


Рис 2. Пример смешанного графа

Родоначальником теории графов считается Леонард Эйлер. В 1736 году в одном из своих писем он формулирует и предлагает решение задачи о семи кёнигсбергских мостах, ставшей впоследствии одной из классических задач теории графов. Термин «граф» впервые ввел Сильвестр, Джеймс Джозеф в 1878 году в своей статье в Nature.

Графы применяются в самых разных областях: при проектировании электрических цепей, телефонных сетей, поиске маршрутов между населенными пунктами, в экономике.

К примеру, в химии обычно с помощью графов молекулы разных калибров: и простые, и сложные. Тесно графы связаны с Интернетом. Благодаря тому, что они когда-то начали идти рука об руку, и произошла цифровая революция.

## Понятие кратчайшего расстояния

Пусть — ребро, соединяющее две вершины: и . Дана весовая функция , которая отображает ребра на их веса, значения которых выражаются действительными числами, и неориентированный граф . Тогда кратчайшим путём из вершины в вершину будет называться путь ), где , который имеет минимальное значение суммы

## Алгоритм Дейкстры

Алгоритм вышел на свет благодаря известному учёному в области компьютерных наук Эдсгеру Вибе Дейкстре. Он позволяет находить кратчайшее расстояние между точками в смешанном графе с неотрицательными весами. Заключается методика в таких действиях:

**Шаг 1**. Происходит инициализация вершин весами: стартовой вершине присваивается 0, а всем остальным – некоторое бесконечно большое число.

**Шаг 2**. Все вершины не выделены.

**Шаг 3**. Стартовая вершина объявляется текущей.

**Шаг 4**. Вес всех невыделенных вершин получается по формуле: *вес* невыделенной вершины - минимальное число из старого веса данной вершины, суммы веса текущей вершины и веса *ребра*, соединяющего текущую вершину с невыделенной.

**Шаг 5**. Среди невыделенных вершин ищется *вершина* с минимальным весом. Если таковая не найдена, то считается, что *вес* всех вершин равен бесконечности и *маршрута* не существует. Следовательно, *выход*. Иначе, текущей становится найденная *вершина*. Она же выделяется.

**Шаг 6**. Если текущей вершиной оказывается конечная, то *путь* найден, и его *вес* есть *вес* конечной вершины.

**Шаг 7**. Переход на шаг 4.

## Библиотека Tkinter или графика на Python

Tkinter – современный фреймворк (framework) языка программирования Python, позволяющий писать программы с оконным интерфейсом. Он кроссплатформенный и может быть использован в большинстве распространённых операционных систем (Windows, Linux, Mac OS X и др.).

Tk является базовым классом любого Tkinter приложения. При создании объекта этого класса запускается интерпретатор tcl/tk и создаётся базовое окно приложения.

Tkinter является событийно-ориентированной библиотекой. В приложениях такого типа имеется главный цикл обработки событий. В Tkinter такой цикл запускается методом mainloop. Для явного выхода из интерпретатора и завершения цикла обработки событий используется метод quit.

В рамках данной курсовой работы нашли применение следующие модули и виджеты библиотеки:

* Canvas – полотно для рисования;
* Button – представление кнопки;
* Label – отвечает за метки в программе;
* Message – служит для вывода сообщений прямо в главное окно;
* Toplevel – отвечает за создание модального окна.

# Проектная часть

## Постановка задачи

Задача состоит в написании программного обеспечения для вычисления кратчайшего расстояния между двумя обозначенными вершинами в смешанном графе. Приложение должно представлять из себя окно с графическим интерфейсом, в котором располагаются следующие функциональные части:

* Полотно для рисования графа;
* Кнопка активации и деактивации режима рисования вершин;
* Кнопка активации и деактивации режима рисования рёбер;
* Кнопка очистки полотна рисования;
* Кнопка активации и деактивации режима вычисления кратчайшего расстояния между выбранными на полотне вершинами графа.

Пользователю должна быть предоставлена возможность нажимать на любую из доступных кнопок и активировать одну из функциональных частей по мере надобности. Ко всему прочему, пользователю должна быть предоставлена возможность назначать текущему создаваемому ребру вес через модальное окно либо отменить создание ребра.

Проектирование и разработка приложения для выполнения операций применительно к набору объектов определенного класса. Необходимо реализовать программное представление графа. Для этого следует выяснить, какие формы его представления существуют.

Граф должен быть реализован на основе предлагаемых языком программирования Python структур данных “словарь”, ”список”, “кортеж”. Для аккуратной упаковки всех стандартных структур следует обратиться к методам ООП, т.е. фактически создать новую структуру данных – класс.

## Структура программного приложения

Программное обеспечение условно можно разделить на три составных модулях:

1. Модуль, включающий служебные классы
2. Модуль, отвечающий за рисование графа в графическом окне приложения;
3. Модуль, отвечающий за реализацию графа как программной единицы, а также алгоритма Дейкстры.

Основные операции с графом происходят в последних двух модулях.

## Модуль, включающий служебные классы

В данном модуле содержатся реализации четырёх классов: первые два из них необходимы для поддержания правильного функционирования программы, другие находят применение в методах рисования графа и в реализации графа как класса.

## Модуль, отвечающий за рисование в графическом окне

При работе с графом используются следующие методы:

1. Методы, активирующие белое полотно Canvas для рисования:

* draw\_nodes(self);
* draw\_edges(self);

1. Методы, рисующие заранее предопределенные графические примитивы: окружность с, заранее заданной в программе, радиусом и линия, у которого направление и длина зависят от расположения на его концах вершин графа на полотне. Сигнатуры следующие:

* draw\_n(self, event)
* draw\_e(self, event)

1. Метод, стирающий полотно рисования и удаляющий все характеристики созданного графа, т.е. делающий его пустым:

* clear(self)

## Модуль, отвечающий за реализацию алгоритма Дейкстры и графа как программной единицы

В этот модуль включены реализация графа как самостоятельной структуры данных с помощью класса и метод – программная реализация алгоритма Дейкстры для решения главной поставленной задачи.

## Проектирование графического интерфейса

В задачи графического интерфейса входят обеспечение максимально хорошего пользовательского опыта, возможности в режиме реального времени позволять пользователю рисовать на белом полотне граф, а затем его исследовать по заранее заданному алгоритму.

Информация (сам граф) будет отображаться в графической нотации – это окружности с обозначенными именами и связями между ним, если они существуют. Первые и вторые будут храниться соответственно в полях реализованного класса Graph: списке nodes и словарях edges, weights.

Пользовательский ввод будет обрабатываться нажатием левой кнопки мыши на кнопки (если это активация какого-то действия) или полотно (если идёт процесс рисования), а также вводом с клавиатуры (если это процесс уточнения веса ребра).

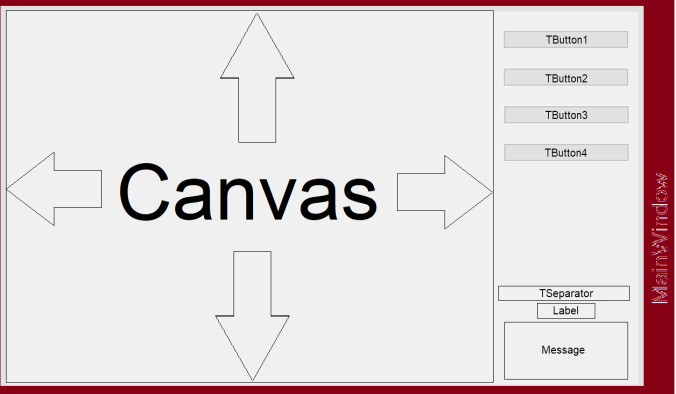


Рис 3. Схематичное изображение интерфейса главного окна

# Реализация и тестирование программного приложения

## **Реализация программного приложения**

## **Хранение деталей о графе в объекте класса Graph**

Для хранения всей информации мной реализован граф как самостоятельная программная единица.

Эта самостоятельная единица есть ни что как класс Graph. Класс Graph реализует сетевую структуру данных. У класса присутствуют 3 атрибута: словарь смешных вершин, словарь весов между смежными вершинами и список всех вершин, и несколько стандартных методов добавления/получения ребер и вершин графа.

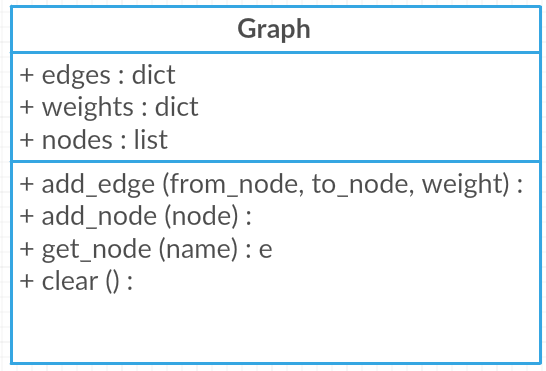


Рис. 4 - Диаграмма класса Graph

В свою очередь, объекты класса Graph впоследствии будут захватывать в себя объекты другого класса. Название этому классу Node. Опишу его. Он представляет из себя вершину графа со следующими атрибутами: наименование, координаты на полотне, соответствующая ей фигура и надпись. Последние 2 атрибута используются в служебных целях. В классе Node, так же как и в Graph, присутствуют стандартные методы добавления/получения полей внешними процессами – их 3.

Другой метод – clear –очищает все сведения о графе, т.е. делает его пустым.

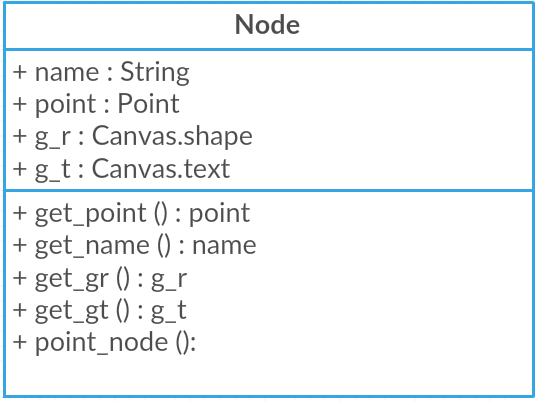


Рис. 5 - Диаграмма класса Node

В свою очередь, Node тоже имеет атрибут непримитивного типа. Речь о point. Данный атрибут является объектом класса Point, задача которого – обеспечить хранение координат вершины на полотне. Получилась такая реализация класса:

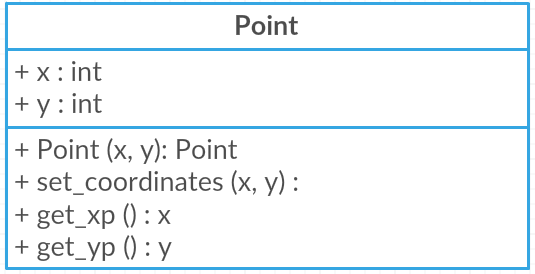


Рис. 6 - Диаграмма класса Point

Класс Point – это два атрибута, обозначающих координаты вершины графа, и методы добавления/получения полей внешними процессами.

## **Реализация алгоритма Дейкстры**

1. Листинг псевдокода:
2. func dijkstra(s):
3. for v∈V
4. d[v] = ∞
5. used[v] = *false*
6. d[s] = 0
7. for i∈Vi
8. v = *null*
9. for j∈V // найдем вершину с минимальным расстоянием
10. if !used[j] and (v == *null* or d[j] < d[v])
11. v = j
12. if d[v] == ∞
13. break
14. used[v] = *true*
15. for e : исходящие из *v* рёбра // произведём релаксацию по всем рёбрам, исходящим из *v*
16. if d[v] + e.len < d[e.to]
17. d[e.to] = d[v] + e.len

2. Блок-схема:

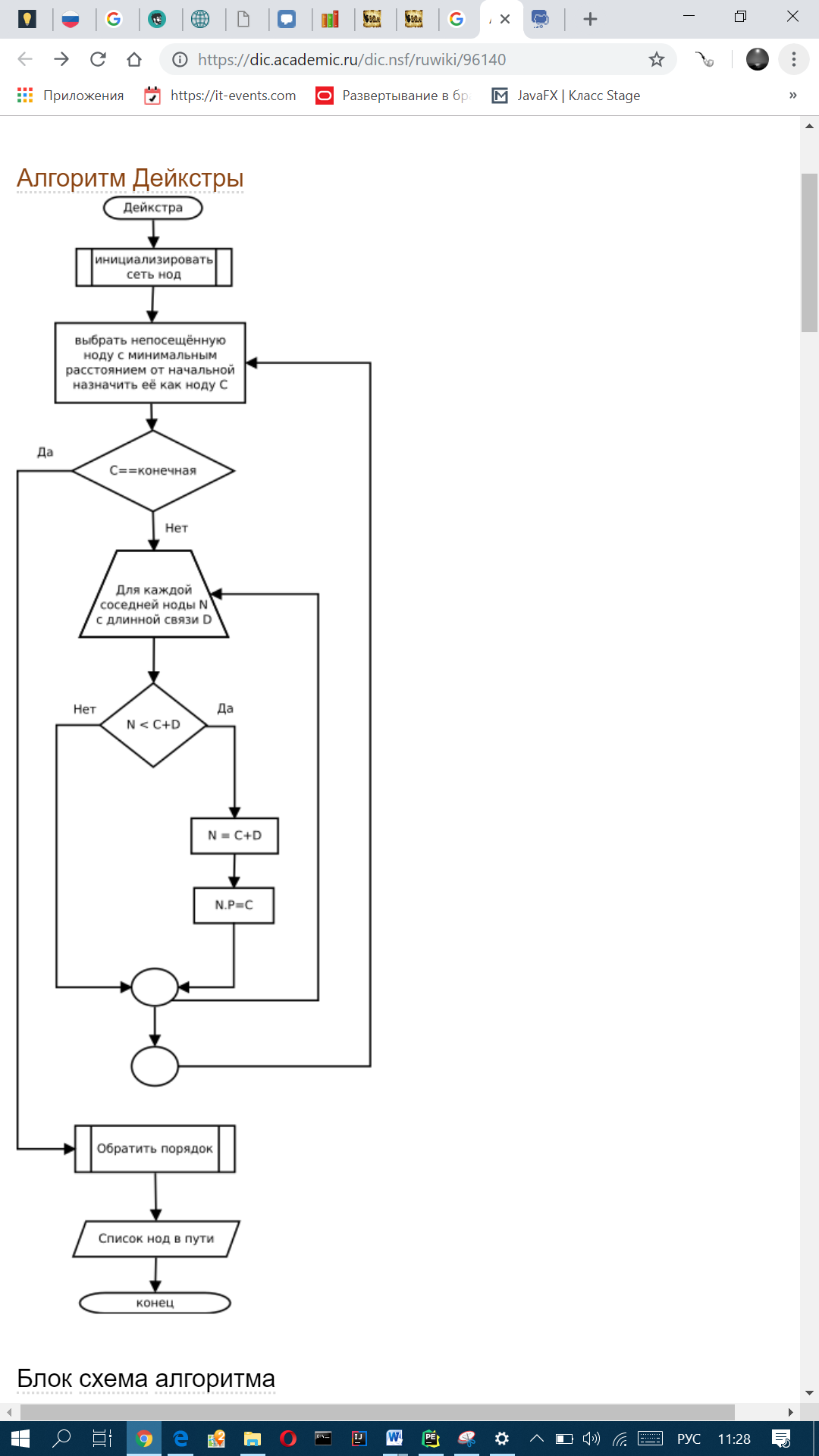


Рис. 7 – Блок-схема алгоритма

## **Реализация проекта приложения**

Проект состоит из 3 файлов:

* system\_class.py – файл реализации класса главного окна приложения, операций с графами (рисование, выделение вершин, удаление графа), служебных классов;
* graph.py – файл реализации класса Graph и алгоритма Дейкстры.
* source\_code.py – файл запуска программы

## **Реализация графического интерфейса**

Приложение имеет графический интерфейс, окно у которого, условно говоря, разделено на две области: кнопки и полотно для рисования графа. Для взаимодействия с интерфейсом пользователю будет достаточно левой кнопки мыши. С помощью нажатий на нее пользователь может взаимодействовать с программой.

Рассмотрим реализацию класса GUI, одного из немногих кроме Graph, который представляет интерес. Класс GUI имеет целое множество атрибутов, среди них:

* 1 локальная копия пустого графа, созданного в основном окне;
* 1 полотно Canvas для рисования;
* 4 кнопки собственной реализации ToggleButton на базе Button;
* 1 разделитель TSeparator;
* 1 информационный вещатель Message;

В последствии в классе также появляется атрибут для обеспечения работы модального окна ввода.

Методов в классе 9:

* draw\_nodes – метод запуска инструмента рисования вершин графа;
* draw\_n – отвечает за рисование одной вершины;
* draw\_edges – метод запуска инструмента рисования рёбер графа;
* draw\_e – отвечает за рисование одного рёбра;
* clear – очищает полотно рисования;
* modal\_win – открывает диалоговое окно ввода веса графа;
* weight – возвращает вес рёбра;
* shortest\_path – метод запуска инструмента, вычисляющего кратчайшее расстояние между двумя вершинами в графе;
* process – метод, запускающий алгоритм Дейкстры.

Полная реализация программного обеспечения содержится в приложении №2.

## **Тестирование ПО**

Таблица 1. - Результаты тестирования программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Описание** | **Ожидаемый результат** | **Полученный результат** | **Вывод** |
| ***1*** | Активация режима создания вершин | Полотно начинает воспринимать касания, регистрировать их и рисовать вершины | Полотно начинает воспринимать касания, регистрировать их и рисовать вершины | Успешно |
| ***2*** | Активация режима проведения рёбер | Полотно начинает воспринимать касания, регистрировать их и связывать необходимые вершины ребрами | Полотно начинает воспринимать касания, регистрировать их и связывать необходимые вершины ребрами | Успешно |
| ***3*** | Очистка полотна | Из полотна исчезают все фигуры и текстовые надписи, удаляется граф | Из полотна исчезают все фигуры и текстовые надписи, удаляется граф | Успешно |
| ***4*** | Нахождение кратчайшего пути | Программа выводит кратчайший путь из пункта А в пункт Б, где А и Б выбираются пользователем из доступных вершин | Программа выводит кратчайший путь из пункта А в пункт Б, где А и Б выбираются пользователем из доступных вершин | Успешно |
| ***5*** | Уточнение веса ребра | Программа получает вес ребра из пункта А в пункт Б (или наоборот) | Программа получает вес ребра из пункта А в пункт Б (или наоборот) | Успешно |
| ***6*** | Уточнение заведомо неправильного веса в программе | Программа игнорирует такой вес | Программа игнорирует такой вес | Успешно |
| ***7*** | Путь из пункта А в пункт А | 0 | 0 | Успешно |
| ***8*** | Путь из А в Б, даже если между ними нет ребра | ∞, т.е. пути не существует | ∞, т.е. пути не существует | Успешно |
| ***9*** | Выход из программы | Закрытие окна программы | Закрытие окна программы | Успешно |

## Тестирование исходного кода на соответствие PEP 8

* Файл graph.py:

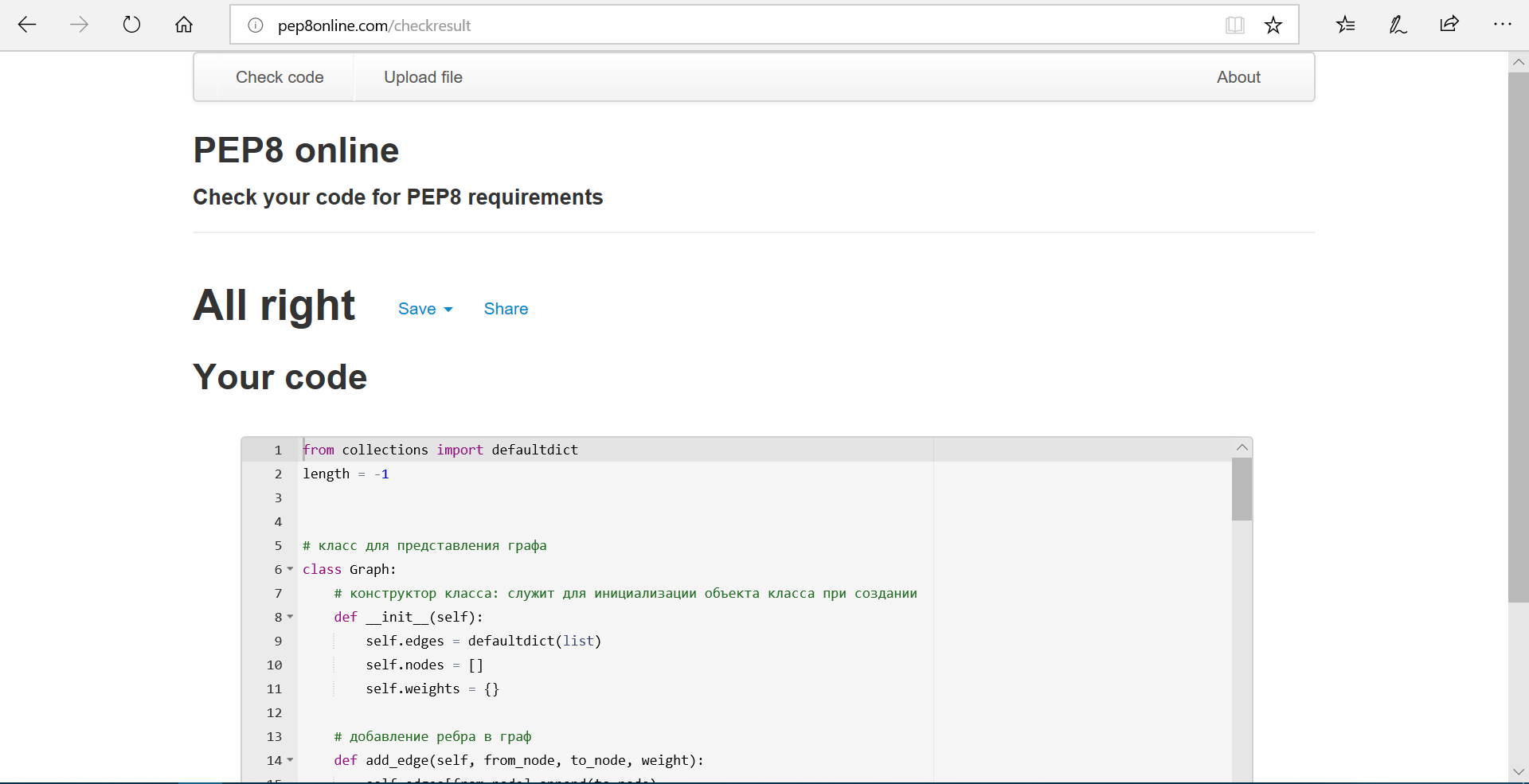


Рис. 8 – Результат тестирования файла graph.py на соответствие спецификации PEP8

* Файл source\_code.py:

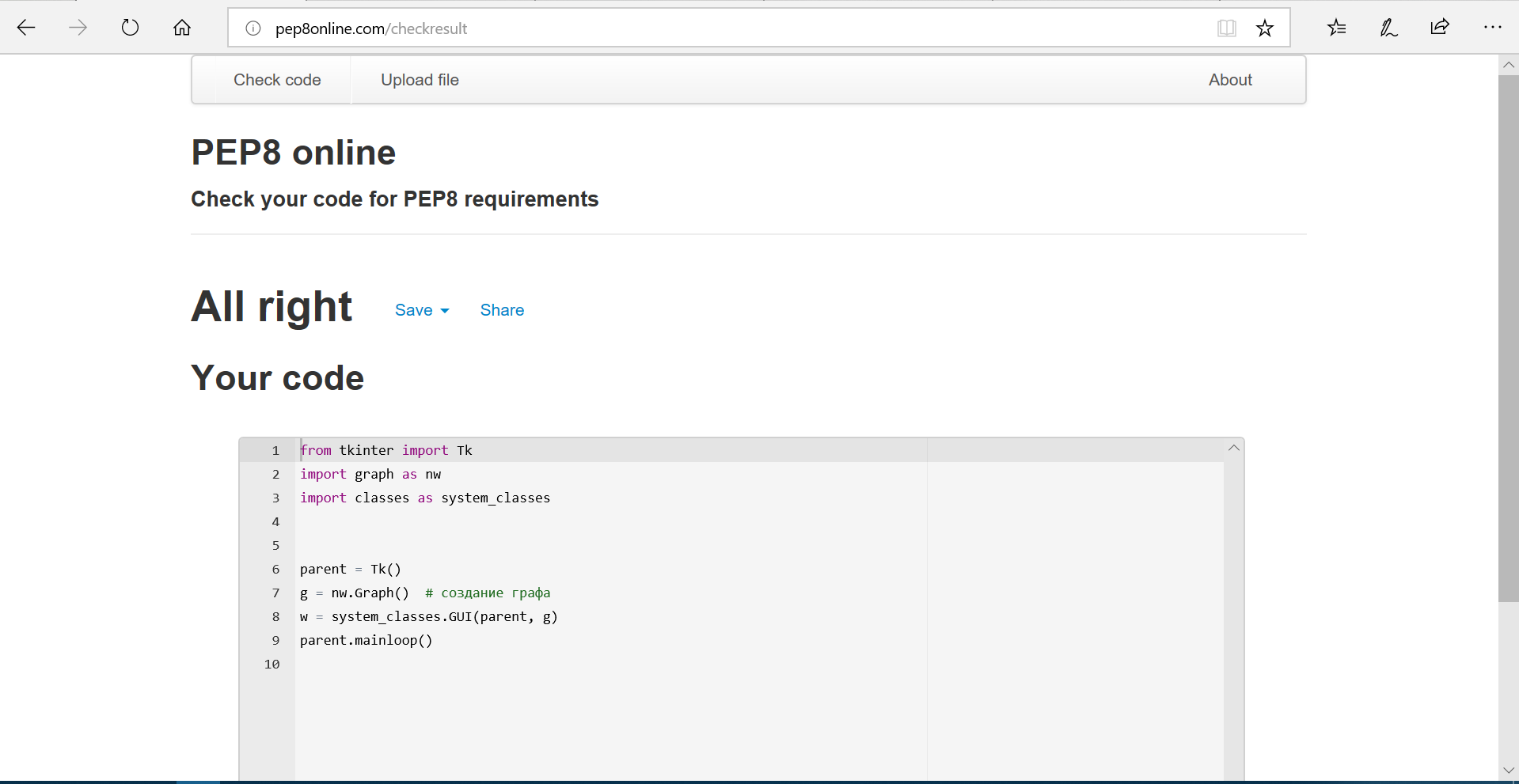


Рис. 9 – Результат тестирования файла source\_code.py на соответствие спецификации PEP8

* Файл classes.py:

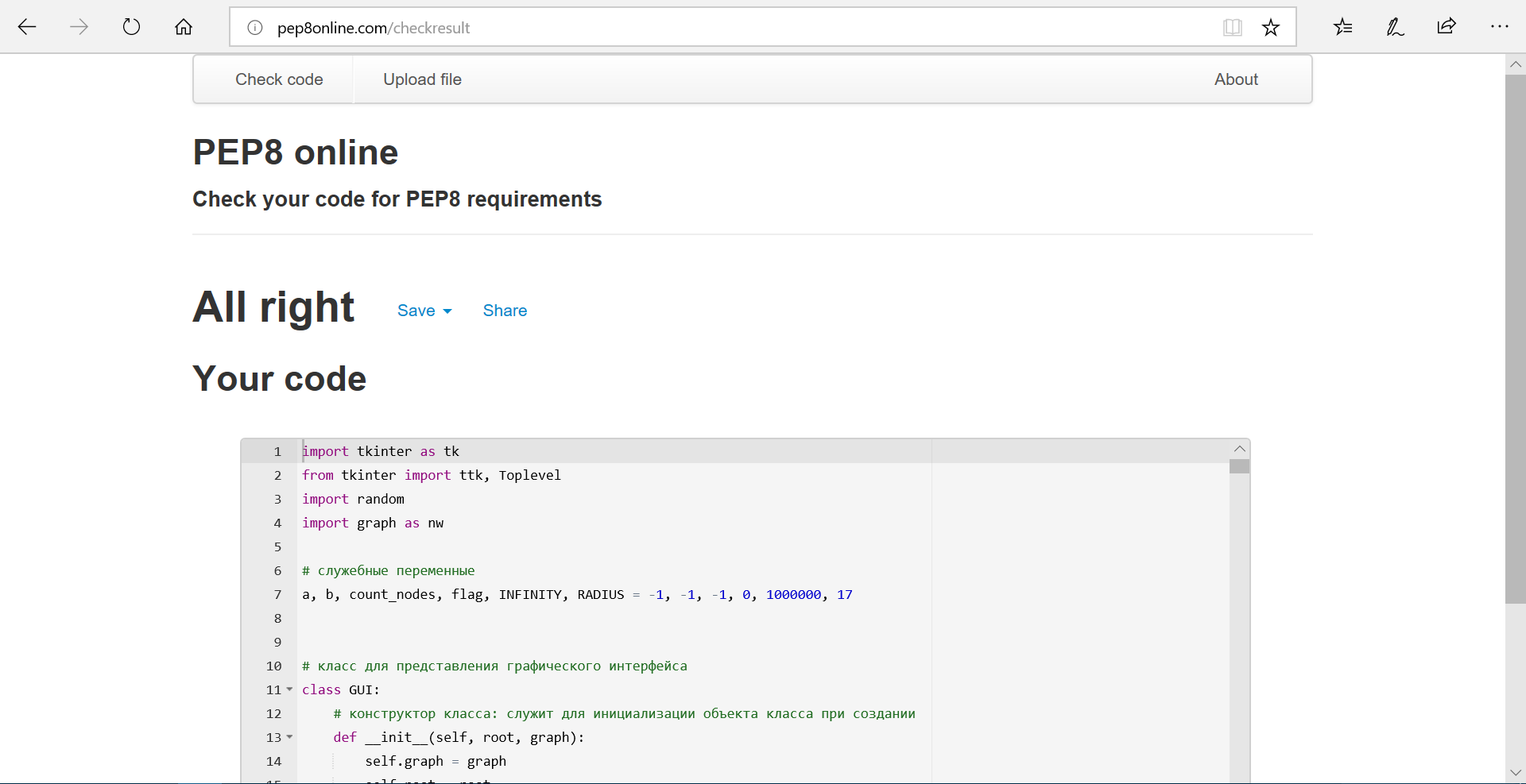


Рис. 10 - Результат тестирования файла classes.py на соответствие спецификации PEP8

# Заключение

Сетевая структура данных -   
пожалуй, самая важная и популярная структура. Не только в плане программирования, но и математики в том числе. Благодаря тому, что графы абстрагируют с математикой, они находят применение везде.

Графы бывают разные. В рамках курсовой работы для вычисления кратчайшего расстояния между двумя вершинами в графе были рассмотрены ориентированные, неориентированные и смешанные графы. Задачу о кратчайшем пути можно поставить по-разному. К примеру, она может быть запросто перенесена на задачу нахождения путей между физическими объектами на таких картографических сервисах, как Карты Google или Яндекс. Карты. Алгоритмов вычисления кратчайших расстояний около десятка. В рамках данной курсовой работы был изучен и применен алгоритм Дейкстры. Это довольно эффективный и повсеместно используемый алгоритм со степенью сложности O(n3). Сами же графы я построил на основе словарей и списков, поскольку посчитал эти структуры данных, предоставляемые языком Python, удобными для выбранного мной алгоритма.

Во время выполнения курсовой работы я узнал принцип действия алгоритма Дейкстры, получил практические навыки по разработке объектно-ориентированных приложений на языке программирования высокого уровня Python, научился создавать графический интерфейс с помощью библиотеки Tkinter и работать с интегрированными средами разработки - Python IDLE и JetBrains PyCharm Edu и понял, как правильно использовать структуры данных на языке Python.

# Список литературы

1. Алгоритмы и структуры данных // NEERC.IFMO,RU: – образовательный ресурс Университета ИТМО [Электронный ресурс] – URL: <https://neerc.ifmo.ru/wiki/index.php?title=Алгоритмы_и_структуры_данных> (дата обращения: 11.11.2018).
2. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ГРАФОВ // MATH.MRSU.RU: – электронный учебник "Экономико-математические методы" [Электронный ресурс] – URL: <http://www.math.mrsu.ru/text/courses/method/osn_pon_teor_graph.htm> (дата обращения: 13.11.2018).
3. Алгоритмы: построение и анализ: учебное пособие по алгоритмам / Клиффорд Штайн, Рональд Линн Ривест, Томас Кормен, и Чарльз Эрик Лейзерсон – Кембридж: MIT Press, 1989. – 1312 c.
4. Сайт автоматизированной проверки программного кода на языке Python на соответствие стандарту PEP 8 // PEP8ONLINE.COM: – PEP 8 Online [Электронный ресурс] – URL: <http://pep8online.com/checkresult> (дата обращения: 15.12.2018).
5. Алгоритмы на графах — Часть 0: Базовые понятия // HABRAHABR.RU: – Хабрахабр [Электронный ресурс] – URL: https://habr.com/post/65367/ (дата обращения: 12.12.2018).

6. Реализация графов и деревьев на Python // HABRAHABR.RU: – Хабрахабр [Электронный ресурс] – URL: https://habr.com/post/112421/ (дата обращения: 17.11.2018).

# Приложения

## Приложение №1. Демонстрация работы приложения

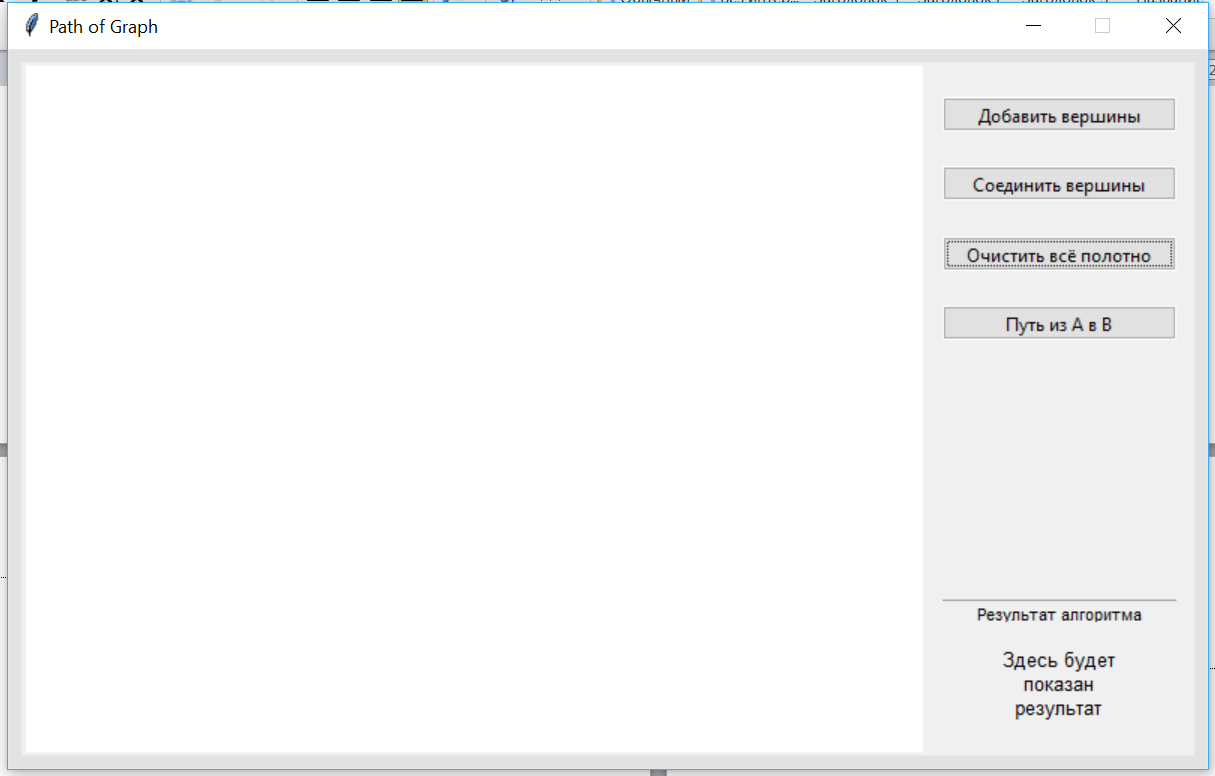


Рис. 11 – Главное окно приложения

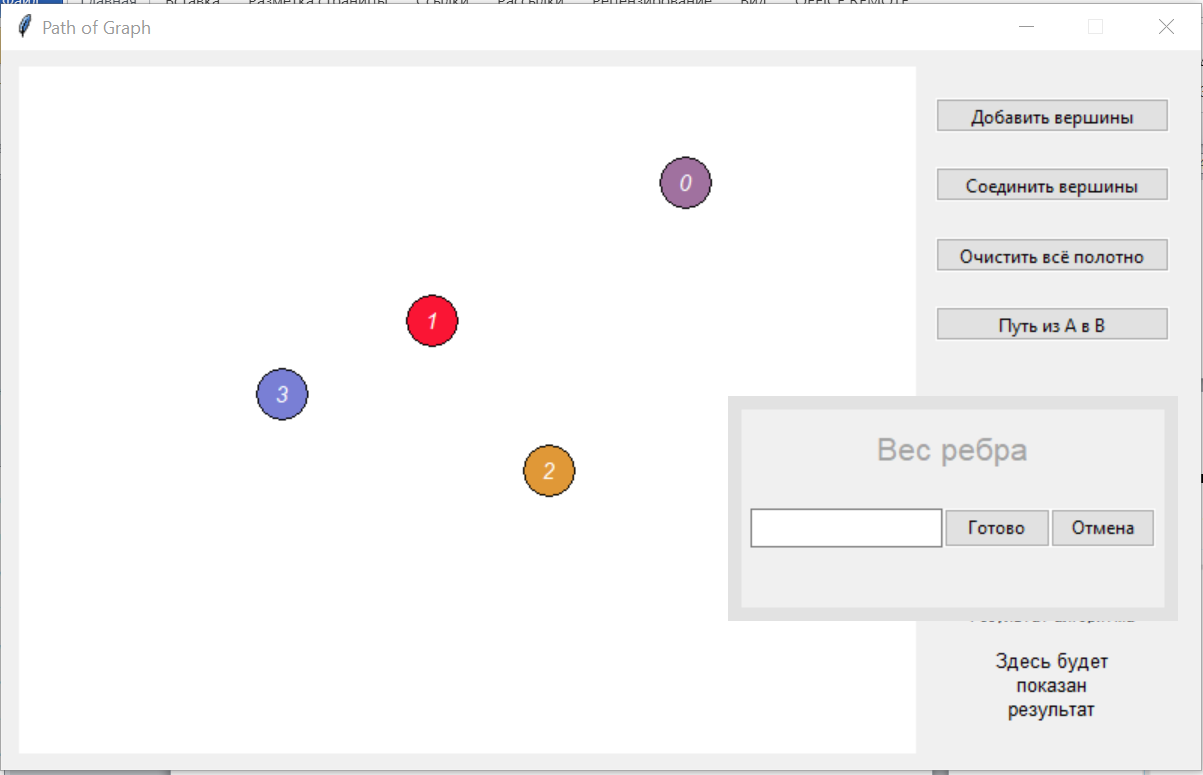


Рис. 12 – Пример рисования графика

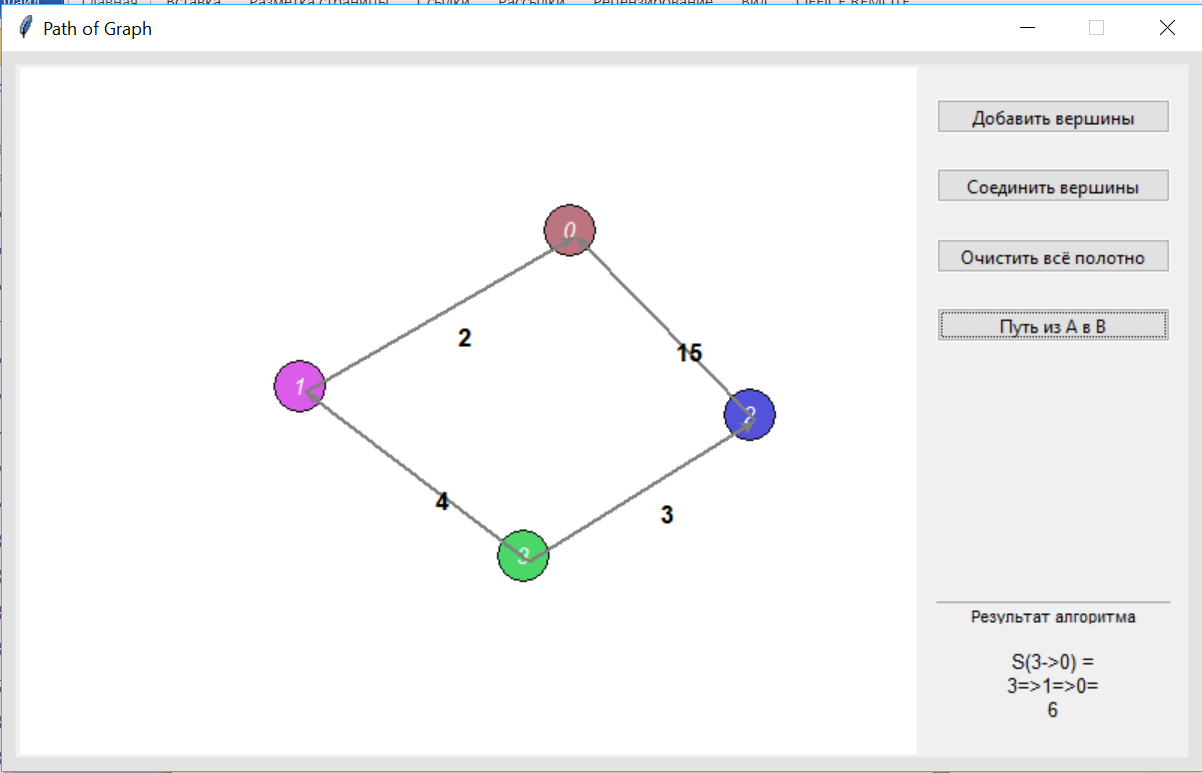


Рис. 13 – Пример нахождения кратчайшего пути между двумя вершинами в графе

## Приложение №2. Листинги исходных текстов программы

* Файл source\_code.py:

1. **from** tkinter **import** Tk
2. **import** graph as nw
3. **import** classes as system\_classes

6. parent = Tk()
7. g = nw.Graph() # создание графа
8. w = system\_classes.GUI(parent, g)
9. parent.mainloop()

* Файл graph.py:

1. **from** collections **import** defaultdict
2. length = -1

5. # класс для представления графа
6. **class** Graph:
7. # конструктор класса: служит для инициализации объекта класса при создании
8. **def** \_\_init\_\_(self):
9. self.edges = defaultdict(list)
10. self.nodes = []
11. self.weights = {}
13. # добавление ребра в граф
14. **def** add\_edge(self, from\_node, to\_node, weight):
15. self.edges[from\_node].append(to\_node)
16. self.weights[(from\_node, to\_node)] = weight
18. # добавление узла в граф
19. **def** add\_node(self, node):
20. self.nodes.append(node)
22. # получение некоторого узла графа по имени
23. **def** get\_node(self, name):
24. **for** e **in** self.nodes:
25. **if** e.get\_name() **is** name:
26. **return** e
28. # очистка графа
29. **def** clear(self):
30. self.edges.clear()
31. self.nodes.clear()
32. self.weights.clear()

35. # алгоритм Дейкстры для смешанных графов
36. **def** algorithm(graph, initial, end):
37. **global** length
38. length = -1
39. shortest\_paths = {initial: (None, 0)}
40. current\_node = initial
41. visited = set()
43. **while** current\_node != end:
44. visited.add(current\_node)
45. destinations = graph.edges[current\_node]
46. weight\_to\_current\_node = shortest\_paths[current\_node][1]
48. **for** next\_node **in** destinations:
49. weight = graph.weights[(current\_node,
50. next\_node)]
51. weight += weight\_to\_current\_node
52. **if** next\_node **not** **in** shortest\_paths:
53. shortest\_paths[next\_node] = (current\_node, weight)
54. **else**:
55. current\_shortest\_weight = shortest\_paths[next\_node][1]
56. **if** current\_shortest\_weight > weight:
57. shortest\_paths[next\_node] = (current\_node, weight)
59. next\_destinations = {node: shortest\_paths[node]
60. **for** node **in** shortest\_paths
61. **if** node **not** **in** visited}
62. **if** **not** next\_destinations:
63. **return** "∞"
64. current\_node = min(next\_destinations,
65. key=**lambda** k: next\_destinations[k][1])
67. path = []
68. **while** current\_node **is** **not** None:
69. path.append(current\_node)
70. next\_node = shortest\_paths[current\_node][0]
71. current\_node = next\_node
72. path = path[::-1]
73. length\_path = list(shortest\_paths[end])
74. length = length\_path[1]
75. **return** path

* Файл graph.py:

1. **import** tkinter as tk
2. **from** tkinter **import** ttk, Toplevel
3. **import** random
4. **import** graph as nw
6. # служебные переменные
7. a, b, count\_nodes, flag, INFINITY, RADIUS = -1, -1, -1, 0, 1000000, 17

10. # класс для представления графического интерфейса
11. **class** GUI:
12. # конструктор класса: служит для инициализации объекта класса при создании
13. **def** \_\_init\_\_(self, root, graph):
14. self.graph = graph
15. self.root = root
16. self.root.resizable(width=False, height=False)
17. self.root.geometry("800x480+200+100")
18. self.root.title("Path of Graph")
19. self.root.configure(highlightbackground="#f0f0f0f0f0f0",
20. highlightcolor="#e2e2e2",
21. highlightthickness="9")
23. self.canvas = tk.Canvas(root)
24. self.canvas.place(relx=0.001, rely=0.001,
25. relheight=0.999, relwidth=0.77)
26. self.canvas.configure(background="#ffffff")
28. self.style = ttk.Style()
29. self.style.configure("TButton",
30. foreground="black",
31. background="white")
32. self.style.map("TButton",
33. foreground=[('disabled', 'white'),
34. ('pressed', 'white'),
35. ('active', 'white'),
36. ('selected', 'black')])
37. self.style.map("TButton",
38. background=[('disabled', 'white'),
39. ('pressed', 'black'),
40. ('active', 'black'),
41. ('selected', '#4a6984')])
42. self.style.map("TButton",
43. relief=[('pressed', 'groove'),
44. ('!pressed', 'ridge')],
45. indicatoron=[('pressed', '#ececec'),
46. ('selected', '#4a6984')])
48. self.TButton1 = ToggleButton(root)
49. self.TButton1.place(relx=0.785, rely=0.05,
50. relheight=0.05, relwidth=0.2)
51. self.TButton1.configure(text='''''Добавить вершины''')
52. self.TButton1.configure(command=self.draw\_nodes)
54. self.TButton2 = ToggleButton(root)
55. self.TButton2.place(relx=0.785, rely=0.15,
56. relheight=0.05, relwidth=0.2)
57. self.TButton2.configure(text='''''Соединить вершины''', takefocus="")
58. self.TButton2.configure(command=self.draw\_edges)
60. self.TButton3 = ToggleButton(root)
61. self.TButton3.place(relx=0.785, rely=0.25,
62. relheight=0.05, relwidth=0.2)
63. self.TButton3.configure(text='''''Очистить всё полотно''', takefocus="")
64. self.TButton3.configure(command=self.clear)
66. self.TButton4 = ToggleButton(root)
67. self.TButton4.place(relx=0.785, rely=0.35,
68. relheight=0.05, relwidth=0.2)
69. self.TButton4.configure(text='''''Путь из А в B''', takefocus="")
70. self.TButton4.configure(command=self.shortest\_path)
72. self.TSeparator1 = ttk.Separator(root)
73. self.TSeparator1.place(relx=0.785,
74. rely=0.774, relwidth=0.2)
76. self.Label = tk.Label(root)
77. self.Label.place(relx=0.785, rely=0.776,
78. relwidth=0.2, relheight=0.04)
79. self.Label.configure(text='Результат алгоритма',
80. foreground="black",
81. font=("Arial", "8"))
83. self.Message = tk.Message(root)
84. self.Message.place(relx=0.785, rely=0.807,
85. relwidth=0.2, relheight=0.18)
86. self.Message.configure(text='Здесь будет показан результат',
87. foreground="black", font=("Arial", "13"),
88. anchor='center', justify='center')
90. # метод запуска инструмента рисования вершин графа
91. **def** draw\_nodes(self):
92. self.TButton1.change\_state()
93. **if** self.TButton1.get\_state() **is** True:
94. **if** self.TButton2.get\_state() **is** True:
95. self.TButton2.change\_state()
96. **if** self.TButton3.get\_state() **is** True:
97. self.TButton3.change\_state()
98. **if** self.TButton4.get\_state() **is** True:
99. self.TButton4.change\_state()
100. self.canvas.bind('<Button-1>', self.draw\_n)
101. **else**:
102. self.canvas.unbind('<Button-1>')
103. **return**
105. # отвечает за рисование одной вершины
106. **def** draw\_n(self, event):
107. **global** count\_nodes
108. count\_nodes += 1
109. x = event.x
110. y = event.y
111. graphic\_shape = self.canvas.create\_oval(x-RADIUS, y-RADIUS,
112. x+RADIUS, y+RADIUS,
113. fill=generate\_color(),
114. outline="black",
115. tag=str(count\_nodes),
116. activefill="#ECDADA")
117. graphic\_text = self.canvas.create\_text(x, y,
118. text=count\_nodes,
119. fill="white",
120. font=("Arial", 11, 'italic'),
121. tag=str(count\_nodes),
122. activefill="white")
123. self.graph.add\_node(Node(str(count\_nodes),
124. x, y, graphic\_shape,
125. graphic\_text))
127. # метод запуска инструмента рисования рёбер графа
128. **def** draw\_edges(self):
129. self.TButton2.change\_state()
130. **if** self.TButton2.get\_state() **is** True:
131. **if** self.TButton1.get\_state() **is** True:
132. self.TButton1.change\_state()
133. **if** self.TButton3.get\_state() **is** True:
134. self.TButton3.change\_state()
135. **if** self.TButton4.get\_state() **is** True:
136. self.TButton4.change\_state()
137. self.canvas.bind('<Button-1>', self.draw\_e)
138. **else**:
139. self.canvas.unbind('<Button-1>')
140. **return**
142. # отвечает за рисование одного ребра графа
143. **def** draw\_e(self, event):
144. **global** flag, a, b
145. **for** e **in** self.graph.nodes:
146. **if** ((e.get\_point().get\_xp() - event.x) \*\* 2 +
147. (e.get\_point().get\_yp() - event.y) \*\* 2) <= (
148. RADIUS \*\* 2):
149. **if** 0 <= flag < 2:
150. flag += 1
151. **if** flag == 1:
152. a = e.get\_name()
153. **else**:
154. b = e.get\_name()
156. **if** (flag == 2) **and** (a **!**= b) **and**\
157. (**not** (b **in** (self.graph.edges.get(a, ())))):
158. self.modal\_win()
159. **if** self.weight() **!=** INFINITY:
160. self.graph.add\_edge(str(a), str(b), self.weight())
161. v1 = self.graph.get\_node(a)
162. v2 = self.graph.get\_node(b)
163. x1 = v1.get\_point().get\_xp()
164. y1 = v1.get\_point().get\_yp()
165. x2 = v2.get\_point().get\_xp()
166. y2 = v2.get\_point().get\_yp()
167. self.canvas.create\_line(x2+4, y2+4,
168. x1+4, y1+4,
169. width=2, fill="grey",
170. smooth=True,
171. activefill="grey",
172. activedash=(5, 4),
173. arrow=tk.FIRST,
174. capstyle=tk.ROUND)
175. x = (x1+x2)/2
176. y = (y1+y2)/2
177. self.canvas.create\_text(x+20, y+20,
178. text=str(self.weight()),
179. fill="black",
180. font=("Arial", "13", 'bold'),
181. activefill="grey")
182. **if** flag == 2:
183. a = b = -1
184. flag = 0
186. # очищает полотно рисования
187. **def** clear(self):
188. self.TButton3.change\_state()
189. **if** self.TButton3.get\_state() **is** True:
190. **if** self.TButton1.get\_state() **is** True:
191. self.TButton1.change\_state()
192. **if** self.TButton2.get\_state() **is** True:
193. self.TButton2.change\_state()
194. **if** self.TButton4.get\_state() **is** True:
195. self.TButton4.change\_state()
196. **global** count\_nodes, a, b, flag
197. a, b, flag, count\_nodes = -1, -1, 0, -1
198. self.canvas.delete("all")
199. self.graph.clear()
200. self.Message.configure(text='Здесь будет показан результат',
201. foreground="black", font=("Arial", "10"),
202. anchor='center', justify='center')
203. **else**:
204. self.canvas.unbind('<Button-1>')
205. **return**
207. # открывает диалоговое окно ввода веса графа
208. **def** modal\_win(self):
209. self.root.dialog = InputDialog(self.root)
211. # возвращает вес ребра
212. **def** weight(self):
213. **return** self.root.dialog.get\_weight()
215. # метод запуска инструмента,
216. # вычисляющего кратчайшее расстояние между двумя вершинами в графе
217. **def** shortest\_path(self):
218. self.TButton4.change\_state()
219. **if** self.TButton4.get\_state() **is** True:
220. **if** self.TButton1.get\_state() **is** True:
221. self.TButton1.change\_state()
222. **if** self.TButton2.get\_state() **is** True:
223. self.TButton2.change\_state()
224. **if** self.TButton3.get\_state() **is** True:
225. self.TButton3.change\_state()
226. self.canvas.bind('<Button-1>', self.process)
227. **else**:
228. self.canvas.unbind('<Button-1>')
229. **return**
231. # метод, запускающий алгоритм Дейкстры
232. **def** process(self, event):
233. **global** flag, a, b
234. **for** e **in** self.graph.nodes:
235. **if** ((e.get\_point().get\_xp() - event.x) \*\* 2 +
236. (e.get\_point().get\_yp() - event.y) \*\* 2) <= (
237. RADIUS \*\* 2):
238. **if** 0 <= flag < 2:
239. flag += 1
240. **if** flag == 1:
241. a = e.get\_name()
242. **else**:
243. b = e.get\_name()
244. **if** flag == 2:
245. path = nw.algorithm(self.graph, a, b)
246. res\_str = "S("+str(a)+"->"+str(b)+") = "
247. **if** nw.length > 0:
248. res\_str += '=>'.join(path) + "=" + str(nw.length)
249. **elif** nw.length **is** 0:
250. res\_str = "S("+str(a)+"->"+str(b)+") = 0"
251. **else**:
252. res\_str += ''.join(path)
253. self.Message.config(text=res\_str, font=("Arial", "10"))
254. a = b = -1
255. flag = 0

258. # класс диалогового окна
259. **class** InputDialog:
260. # конструктор класса: служит для инициализации объекта класса при создании
261. **def** \_\_init\_\_(self, root):
262. self.weight = INFINITY
263. self.slave = Toplevel(root)
264. self.slave.overrideredirect(1)
265. self.slave.resizable(width=False, height=False)
266. self.slave.geometry("300x150+500+300")
267. self.slave.configure(highlightbackground="#f0f0f0f0f0f0",
268. highlightcolor="#e2e2e2",
269. highlightthickness="9")
271. self.Label = tk.Label(self.slave)
272. self.Label.place(relx=0.001, rely=0.1, relwidth=0.999, relheight=0.2)
273. self.Label.configure(text='Вес ребра',
274. background="#f0f0f0", foreground="#a3a3a3",
275. font=("Arial", "16"))
277. self.TEntry = ttk.Entry(self.slave)
278. self.TEntry.place(relx=0.02, rely=0.5, relheight=0.2, relwidth=0.455)
280. self.TButton1 = ttk.Button(self.slave)
281. self.TButton1.place(relx=0.48, rely=0.5,
282. relheight=0.2, relwidth=0.250)
283. self.TButton1.configure(text='Готово', takefocus="",
284. command=self.set\_weight)
286. self.TButton2 = ttk.Button(self.slave)
287. self.TButton2.place(relx=0.73, rely=0.5,
288. relheight=0.2, relwidth=0.250)
289. self.TButton2.configure(text='Отмена', takefocus="",
290. command=self.set\_inf)
292. self.slave.grab\_set()
293. self.slave.focus\_set()
294. self.slave.wait\_window()
296. # проверяет, удовлетворяет ли введенный вес
297. # минимальным требованиям и если да, то закрывает диаологовое окно ввода
298. **def** set\_weight(self):
299. **try**:
300. self.weight = int(self.TEntry.get())
301. **if** self.weight > 0:
302. self.slave.destroy()
303. **except** ValueError:
304. **pass**
306. # позволяет отменить рисование ребра между выбранными вершинами
307. **def** set\_inf(self):
308. self.weight = INFINITY
309. self.slave.destroy()
311. # служит для возвращения полученного веса ребра за пределы диалогового окна ввода
312. **def** get\_weight(self):
313. **return** self.weight

316. # класс для представления одной вершины графа
317. **class** Node:
318. **def** \_\_init\_\_(self, name, x, y, g\_r=None, g\_t=None):
319. self.name = name
320. self.point = Point(x, y)
321. self.g\_r = g\_r
322. self.g\_t = g\_t
324. **def** get\_point(self):
325. **return** self.point
327. **def** get\_name(self):
328. **return** self.name
330. **def** get\_gr(self):
331. **return** self.g\_r
333. **def** get\_gt(self):
334. **return** self.g\_t
336. **def** print\_node(self):
337. **print**(str(self.name) + ": " +
338. str(self.point.x) + ', ' +
339. str(self.point.y))

342. # класс для представления кнопки-переключателя
343. **class** ToggleButton(ttk.Button):
344. **def** \_\_init\_\_(self, relative=None, state=False):
345. ttk.Button.\_\_init\_\_(self, relative)
346. self.state = state
348. **def** change\_state(self):
349. self.state = **not** self.state
350. **return** self.state
352. **def** get\_state(self):
353. **return** self.state

356. # класс для представления точки на экране
357. **class** Point:
358. **def** \_\_init\_\_(self, x=0, y=0):
359. self.x = x
360. self.y = y
362. **def** set\_coordinates(self, x, y):
363. self.x = x
364. self.y = y
366. **def** get\_xp(self):
367. **return** self.x
369. **def** get\_yp(self):
370. **return** self.y

373. # генератор случайного цвета
374. **def** generate\_color():
375. color = '#{:02x}' \
376. '{:02x}' \
377. '{:02x}'.format(\*map(**lambda** x: random.randint(0, 255), range(3)))
378. **return** color

## Приложение №3. Скриншоты с презентации к курсовой работе

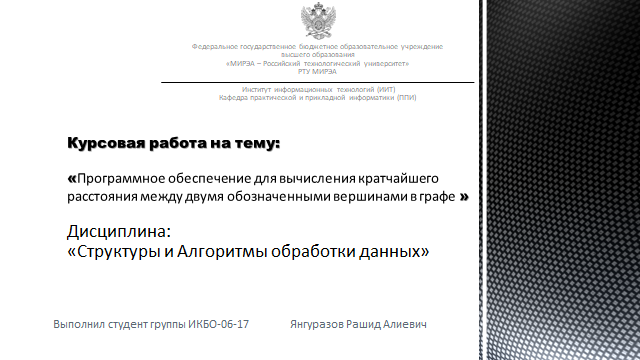


Рис. 14 – Слайд 1

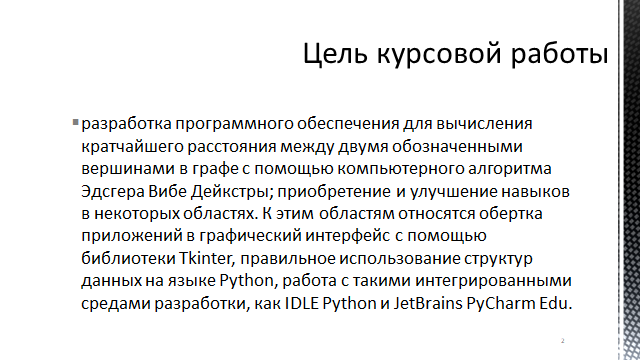
\

Рис. 15 – Слайд 2

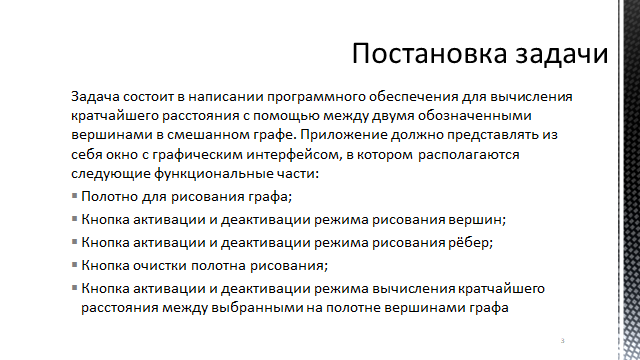


Рис. 16 – Слайд 3

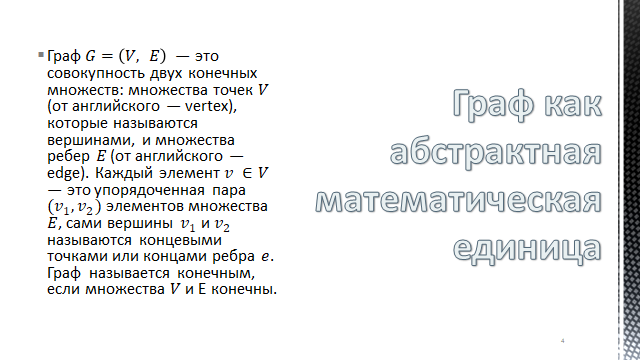


Рис. 17 – Слайд 4

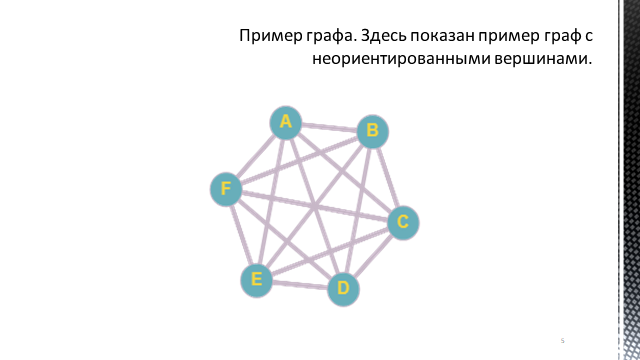


Рис. 18 – Слайд 5

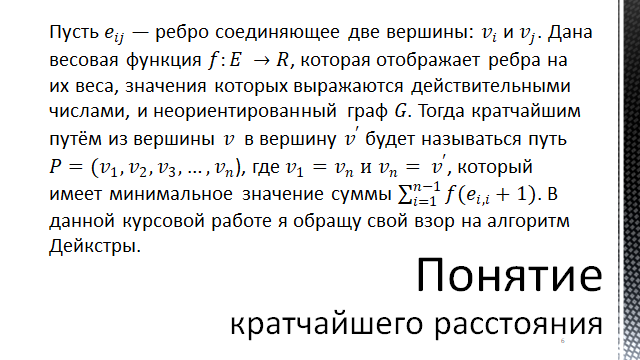


Рис. 19 – Слайд 6

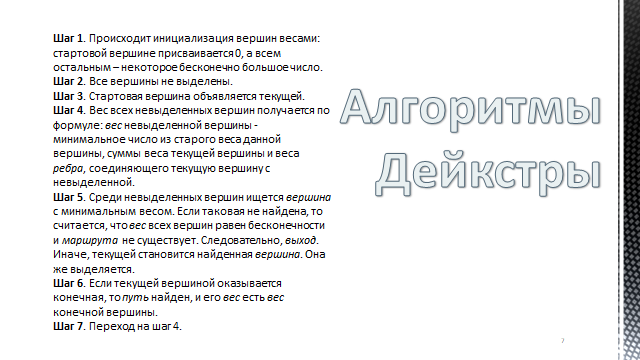


Рис. 20 – Слайд 7



Рис. 21 – Слайд 8

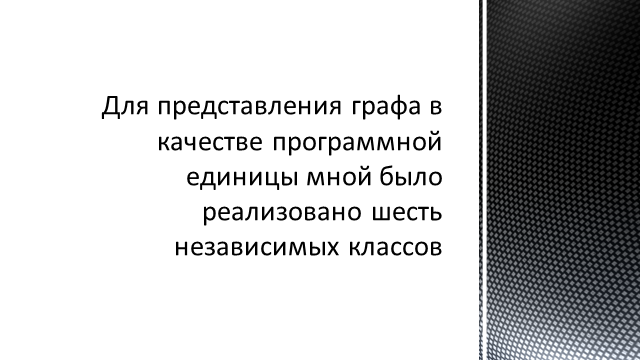


Рис. 22 – Слайд 9

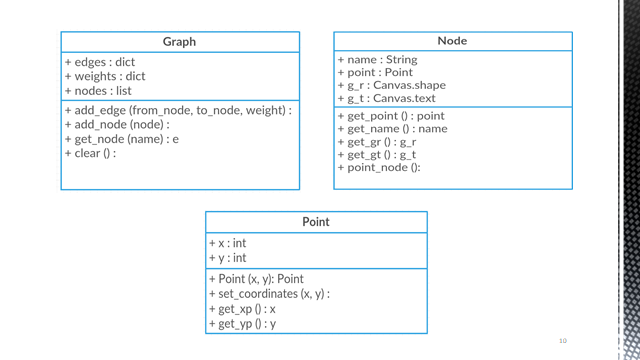


Рис. 23 – Слайд 10

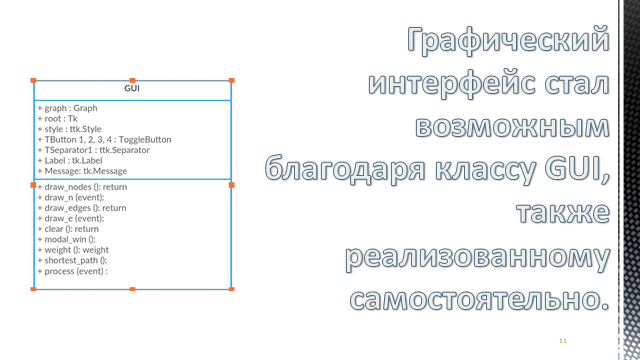


Рис. 24 – Слайд 11

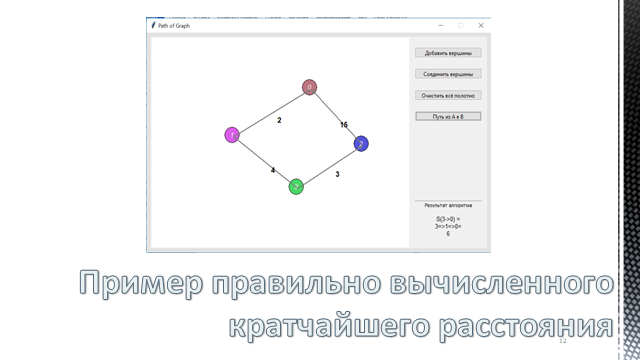


Рис. 25 – Слайд 12