МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

Институт компьютерных наук

Кафедра Инженерной Кибернетики

**Курсовая работа**

по дисциплине

«Алгоритмы дискретной математики» на тему

«Задача о кратчайшем пути (ЗКП) и ее решение методом динамического

программирования»

Выполнили:

студенты 2-го курса,

гр. БПМ-22-1 Абрамов И. А.

гр. БПМ-22-1 Каневский Д. Е.

гр. БПМ-22-1 Махров М. А.

гр. БПМ-22-1 Нургалиев Н. Д.

гр. БПМ-22-1 Труфманов М. А.

Проверил:

доцент, Пышняк М. О.

Москва, 2024

**Оглавление**

Глава 1. Введение…………………………………………………………………3

Глава 2. Аналитический разбор литературы…………………………………….5

Глава 3. Предметная область……………………………………………………..8

Глава 4. Содержательная постановка задачи……………………………………9

Глава 5. Математическая постановка задачи……………………………………9

Глава 6. Основная часть…………………………………………………………10

Глава 7. Алгоритмы……………………………………………………………...22

Глава 8. Список литературы…………………………………………………….25

**Введение**

*Актуальность выбранной темы курсовой работы:*

Задача о поиске кратчайшего пути на графе остаётся актуальной и важной во многих областях, таких как транспортная логистика, сетевое планирование, маршрутизация в компьютерных сетях, анализ социальных сетей и многие другие.

Алгоритм Дейкстры, а также его модификации используются в области сетей, транспортных потоков, и, конечно же, в области обработки графов. Например, протокол динамической маршрутизации OSPF разбивает процесс построения таблицы маршрутизации на два этапа, и второй этап состоит в нахождении оптимальных маршрутов с помощью созданного на первом этапе графа. На этом этапе применяется итеративный алгоритм Дейкстры: каждый маршрутизатор считает себя центром сети и ищет оптимальный маршрут до каждой известной ему сети. В каждом найденном таким образом маршруте запоминается только один шаг - до следующего маршрутизатора. Данные об этом шаге попадают в таблицу маршрутизации. Если несколько маршрутов имеют одинаковую метрику до сети назначения, то в таблице маршрутизации запоминаются первые шаги всех этих маршрутов. Также алгоритм применяется при эвакуации населения из очагов бедствия. Оптимальные маршруты до пунктов сбора транспорта для каждой группы людей в штабе МЧС рассчитывает программа на основе алгоритма Дейкстры.

|  |
| --- |
|  |

В области разработки игр широкое применение находит алгоритм А\*, классическим примером его применения является игра «Lines», «ColorLines», в которой игроку необходимо составить в ряд несколько шаров, путь шара как раз рассчитывается с помощью этого алгоритма. Кроме этого, А\* применяется во многих играх жанра RTS (Real Time Strategy), его модификации применяются в таких крупных проектах как «StarCraft2» и «Warcraft3».

*Мотивация реализации темы курсовой работы:*

1. Кратчайшие пути играют важную роль в различных прикладных областях, таких как транспортная логистика, маршрутизация пакетов в компьютерных сетях, планирование маршрутов для роботов и автономных транспортных средств, а также в GPS-навигации и сетевом проектировании.
2. Решение задачи о кратчайшем пути приводит к разработке и оптимизации различных алгоритмов, таких как алгоритм Дейкстры, алгоритм Беллмана-Форда и алгоритм A\*, которые находят применение не только в поиске кратчайших путей, но и в других задачах.
3. Задача о кратчайшем пути привлекает внимание теоретиков и практиков, которые исследуют различные аспекты её решения, включая сложность алгоритмов, поиск оптимальных стратегий и разработку новых методов решения.
4. Поиск кратчайших путей может быть сложной задачей, особенно на больших и сложных графах. Разработка эффективных алгоритмов для решения этой задачи является вызовом для исследователей и инженеров.

В анализе социальных сетей, транспортных потоков, биологических сетей и других областях кратчайшие пути используются для изучения структуры и связей между элементами.

**Аналитический обзор литературы**

Существует несколько алгоритмов поиска кратчайшего пути в графах. Некоторые из них включают в себя:

1. **Алгоритм Дейкстры (Dijkstra's Algorithm)**: используется для нахождения кратчайшего пути от одной вершины до всех остальных взвешенных графах без отрицательных ребер.
2. **Алгоритм A\* (A-star Algorithm)**: эффективно находит кратчайший путь между двумя вершинами в графе с весами ребер, используя эвристику для ускорения поиска.
3. **Алгоритм Беллмана-Форда (Bellman-Ford Algorithm)**: работает с графами с отрицательными весами ребер и находит кратчайший путь между вершинами, даже если есть отрицательные циклы.
4. **Алгоритм Флойда-Уоршелла (Floyd-Warshall Algorithm)**: находит кратчайшие пути между всеми парами вершин во взвешенном графе, даже с отрицательными весами и циклами.
5. **Алгоритм Джонсона (Johnson's Algorithm)**: решает проблему поиска кратчайших путей для всех пар вершин в графе, включая графы с отрицательными весами.

Рассмотрим истории и сравнения некоторых из них:

*История*

1. **Алгоритм Дейкстры** был разработан голландским ученым Эдсгером Дейкстрой в 1959 году. Интересно отметить, что разработка этого алгоритма произошла в контексте его работы над системой программирования для ЭВМ EDSAC (Electronic Delay Storage Automatic Calculator) в Университете Кембриджа.  
   В то время, когда Дейкстра работал над этой системой, задача нахождения кратчайших путей в графах стала важной проблемой, особенно в контексте маршрутизации в телекоммуникационных сетях. Ученый сталкивался с необходимостью эффективного поиска оптимальных маршрутов, что послужило стимулом к созданию алгоритма.  
   Впервые алгоритм был описан в письменной форме в статье Дейкстры «A Note on Two Problems in Connexion with Graphs» («Замечание по двум проблемам, связанным с графами»), опубликованной в журнале Numerische Mathematik в 1959 году. В этой статье ученый представил алгоритм, который стал известен как алгоритм Дейкстры, предназначенный для нахождения кратчайших путей в графах с неотрицательными весами ребер.  
   Алгоритм Дейкстры стал важным вкладом в область комбинаторной оптимизации и теории графов, а его применение нашло широкое распространение в различных областях, таких как сетевые технологии, транспортная логистика и многие другие.
2. История **алгоритма Беллмана-Форда** связана сразу с тремя независимыми математиками: Лестером Фордом, Ричардом Беллманом и Эдвардом Муром. Форд и Беллман опубликовали алгоритм в 1956 и 1958 годах соответственно, а Мур сделал это в 1957 году. И иногда его называют алгоритмом Беллмана – Форда – Мура.

*Сравнение*

Проанализировав реализацию алгоритмов Дейкстры и Беллмана-Форда, мы выявили несколько отличий (табл.1):

|  |  |  |
| --- | --- | --- |
|  | **Алгоритм Дейкстры** | **Алгоритм Беллмана-Форда** |
| **Неотрицательные веса** | Корректно работает для ориентированных и неориентированных графов | Корректно работает для ориентированных и неориентированных графов |
| **Отрицательные веса** | Неудачи | Корректно работает только с ориентированными графами |
| **Отрицательные циклы** | Неудачи | Может обнаруживать отрицательные циклы в ориентированных графах |
| **Временная сложность** |  |  |

Таблица 1. Сравнение алгоритмов Дейкстры и Беллмана-Форда

**Вывод**: из таблицы 1 видно, что алгоритм Дейкстры лучше, когда дело доходит до сокращения временной сложности. Однако, когда отрицательные веса, нужно использовать алгоритм Беллмана-Форда. Кроме того, если задача – узнать, содержит ли график отрицательные циклы или нет, алгоритм Беллмана-Форда может помочь в этом. В случае отрицательных весов или даже отрицательных циклов алгоритм Беллмана-Форда работает только с ориентированными графами.

**Предметная область**

В работе будут рассмотрены основные понятия задачи о кратчайшем пути в графах, такие как понятие пути, вес ребра, длина пути, а также различные подходы к ее решению.

**граф** — это множество точек (вершин) и простых кривых (ребер), т. е. порядок (количество вершин), размерность (количество ребер.) Замкнутое ребро (петля), направленное ребро (дуга). Аналог **дуги** - одностороннее шоссе или река с течением. Пусть . Каждой дуге с номером, где (разность номеров вершин) припишем ее длину . **Путем (маршрутом)** назовем в графе конечную последовательность вершин и дуг (ребер) ;  длина пути из вершины в .

**Вершина** - точка в графе, отдельный объект, для топологической модели графа не имеет значения координата вершины, её расположение; однако при решении некоторых задачах вершины могут раскрашиваться в разные цвета или сохранять числовые значения.

**Ребро** - неупорядоченная пара двух вершин, которые связаны друг с другом.

**Вес ребра** в графе – это числовая характеристика, которая присваивается каждому ребру в графе. Он представляет собой значение или метку, которая указывает на стоимость, длину, время или другую меру, связанную с этим ребром.

**Взвешенный граф** - граф, в котором у каждого ребра и/или каждой вершины есть “вес” - некоторое число, которое может обозначать длину пути, его стоимость и т. п.

**Длина пути** — это число рёбер, используемых в пути, при этом многократно используемые рёбра считаются соответствующее число раз. Длина может равняться нулю, если путь состоит только из одной вершины.

**Цикл** или **Контур** - цепь, в котором последняя вершина совпадает с первой.

**Содержательная постановка задачи**

1. **Описание графа**: это может быть ориентированный или неориентированный граф, представляющий собой сеть узлов (вершин) и связей между ними (рёбер).
2. **Начальная и конечная вершины**: при генерации графа нужно указать начальную и конечную вершины, относительно которых будет считаться кратчайший путь.
3. **Вес рёбер**: указывается при создании графа. Программа может работать как с взвешенными графами, так и с невзвешенными.
4. **Алгоритмы**: разработка методов для поиска кратчайшего пути в сгенерированном графе.

**Математическая постановка задачи**

1. **Граф**. Это абстрактное представление сети путей, где узлы представляют местоположения, а рёбра - пути между этими местоположениями. Граф может быть ориентированным или неориентированным.
2. **Вершины и рёбра графа**. Вершины представляют местоположения, а рёбра представляют собой связи между этими местоположениями. Каждое ребро может иметь вес, представляющий стоимость перемещения между вершинами или расстояние между ними.
3. **Функция стоимости**. Это функция, которая определяет стоимость или вес каждого ребра в графе.
4. **Начальная и конечная вершины**. Это вершины, между которыми необходимо найти кратчайший путь.
5. **Целевая функция**. Это функция (алгоритм), по которому ищется кратчайший путь между двумя вершинами в графе.
6. **Ограничения.** Веса рёбер и циклы не могут быть отрицательными.

**Основная часть**

*Практическая часть проекта*

GitHub репозиторий проекта (вместе с документацией и прописанным README.md файлом): <https://github.com/Mihail20052005/adm_course_work>

В рамках выполнения курсовой работы было написано приложение на языке Python с разработанным GUI (PyQt5). Для визуализации работы алгоритмов был написан модуль, создающий изображение графа и окрашивающий кратчайший путь в нем (Networkx, matplotlib)

Был разработан класс Graph предоставляющий функциональность для работы с графами.

В его задачи входит создание графа, генерация случайного графа, добавление новых элементов (вершин), сохранение в файл в формате .png изображения.

Разберем подробнее каждую структуру, используемую в проекте.

*Директории (рис. 1):*

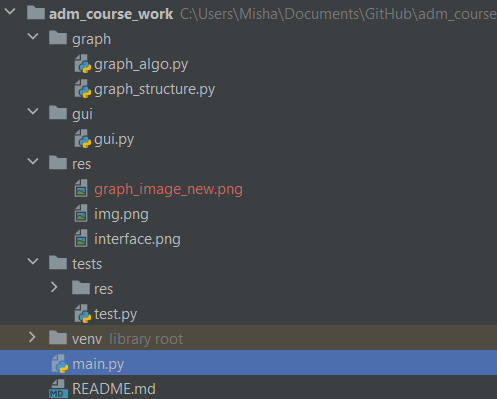


Рисунок 1. Директории проекта

1. graph – структура графа и методы для работы с ним.
   1. graph/graph\_algo.py – алгоритмы (Дейкстра, Беллман-Форда), поиск кратчайшего пути в графе, распечатывание кратчайшего пути в графе.
   2. graph/graph\_structure – описана структура графа, возможности его создания и отображения.
2. gui – создание и использование GUI.
3. res – ресурсы для работы программы (сохраненные изображения).
4. tests – директория с тестами, написанными для проверки корректности работы алгоритмов.

Разберем подробно каждый из методов, используемых в проекте

*Методы graph\_structure.py*

1. **def \_\_init\_\_ (self, mode)** (рис. 2)– конструктор класса Graph. Создает поля self.graph (словарь точек и весов) и self.mode – выбор алгоритма поиска кратчайшего пути.   
   **Args: mode (str)** – Режим выполнения.

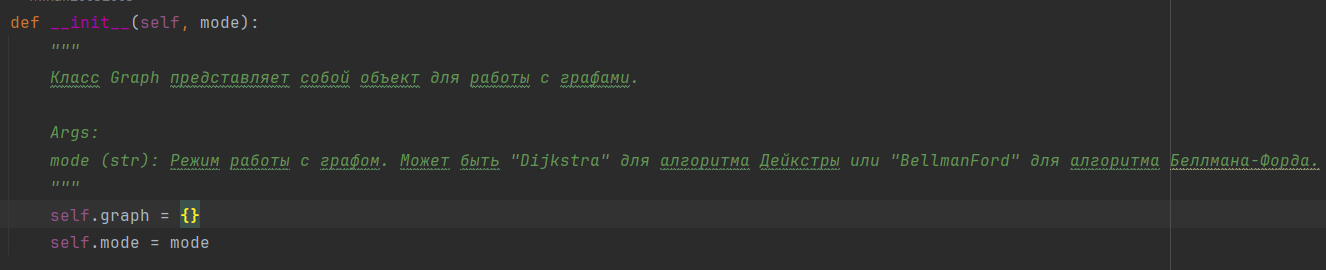


Рисунок 2. Метод def \_\_init\_\_

1. **def graph\_(self)** (рис. 3) – метод возвращающий словарь, представляющий граф. Создает поля self.graph (словарь точек и весов) и self.mode – выбор алгоритма поиска кратчайшего пути.

**Args**: -

**Returns: dict**: словарь, где ключи – это узлы, а значения смежных узлов с их весами.

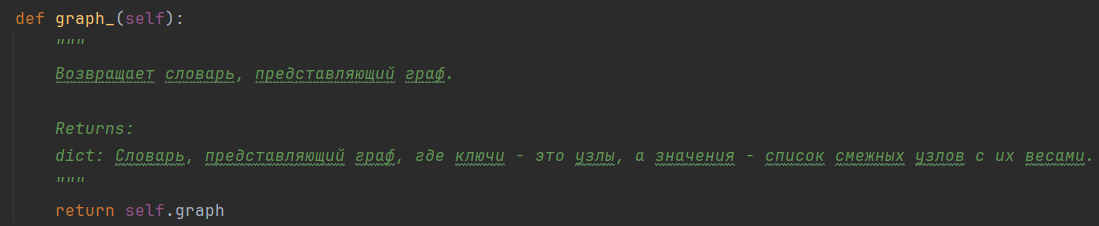


Рисунок 3. Метод def graph\_(self)

1. **def add\_edge(self, node1, node2, weight)** (рис. 4) – метод добавляющий ребро между двумя узлами в граф.

**Args: node1, node2** – узлы, **weight** – вес ребра между узлами.

**Описание работы алгоритма:**

1. Проверяется, есть ли узел **node1** в словаре **self.graph**. Если нет, то создается пустой список смежных узлов для **node1**.
2. Затем кортеж (**node2**, **weight**) добавляется в список смежных узлов **node1**. Этот кортеж представляет собой информацию о смежном узле **node2** и весе ребра между **node1** и **node2**.
3. Аналогично проверяется, есть ли узел **node2** в словаре **self.graph**. Если нет, то создается пустой список смежных узлов для **node2**.
4. Затем кортеж (**node1**, **weight**) добавляется в список смежных узлов **node2**. Этот кортеж представляет собой информацию о смежном узле **node1** и весе ребра между **node2** и **node1**.

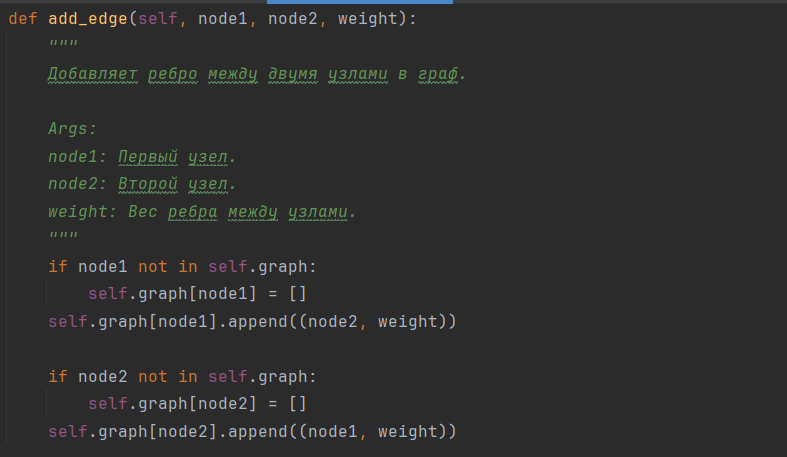


Рисунок 4. Метод def add\_edge

1. **def generate\_random\_graph(self, num\_nodes, num\_edges)** (рис. 5) – генерирует случайный граф

**Args: num\_nodes** – количество узлов в графе, **num\_edges** – количество ребер в графе

**Описание работы алгоритма:**

1. Создаются пустые списки смежных узлов для каждого узла от 1 до **num\_nodes**.
2. Создается список всех возможных рёбер в графе.
3. После генерации всех возможных рёбер список **edges** случайным образом перемешивается
4. Затем выбираются первые **num\_edges** рёбер из перемешанного списка **edges**, и для каждого выбранного ребра вызывается метод **add\_edge**, чтобы добавить его в граф с весом 1.

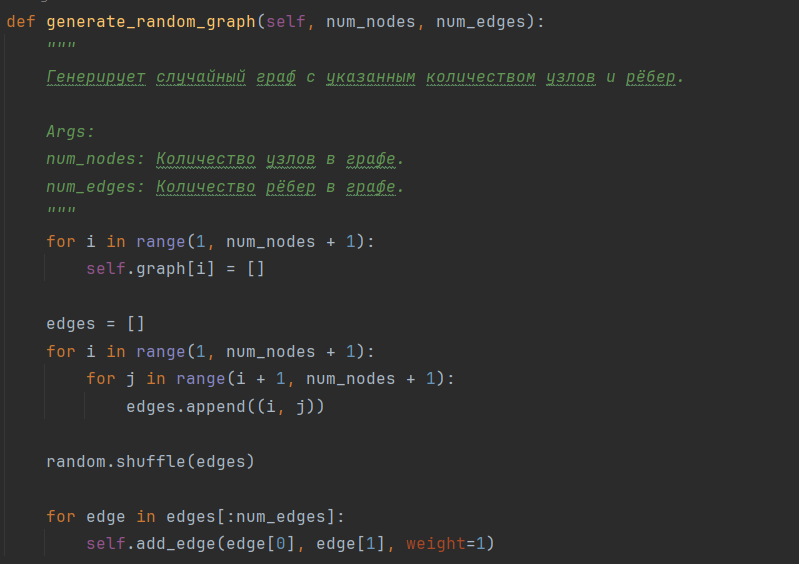


Рисунок 5. Метод def generate\_random\_graph

1. **def save\_graph (self, num\_nodes, num\_edges)** (рис. 6) – сохраняет изображение графа ф изображенным красным цветом кратчайшим пути в нем.

**Args: source** – исходный узел в графе **target** – узел до которого необходимо дойти.

**Описание работы алгоритма:**

1. Создается новый объект графа G из библиотеки NetworkX (**nx.Graph()**).
2. Вызывается функция **find\_shortest\_path** из модуля **graph\_algo**, чтобы найти кратчайший путь между заданными узлами **source** и **target** в текущем графе **self.graph**. Эта функция возвращает кортеж, где первый элемент - кратчайший путь, а второй элемент - его длина. Мы интересуемся только самим путем, поэтому \_ используется для игнорирования второго элемента.
3. Затем для каждого узла в графе **self.graph** и его соседей выполняется добавление рёбер в объект **G** с весами. Это делается с помощью вложенных циклов **for**, где для каждого узла и его соседей вызывается **G.add\_edge(node, neighbor, weight=weight)**.
4. Позиции узлов на графическом изображении определяются с помощью алгоритма распределения узлов в сетке **spring\_layout**. После этого вызывается **nx.draw**, чтобы нарисовать граф с помощью библиотеки **matplotlib**.
5. Затем узлы, через которые проходит кратчайший путь, выделяются красным цветом с помощью **nx.draw\_networkx\_nodes**.
6. Устанавливается заголовок графа с помощью **plt.title**.
7. Граф сохраняется в файл "**res/graph\_image\_new.png**" с разрешением 100 dpi с помощью **plt.savefig**.

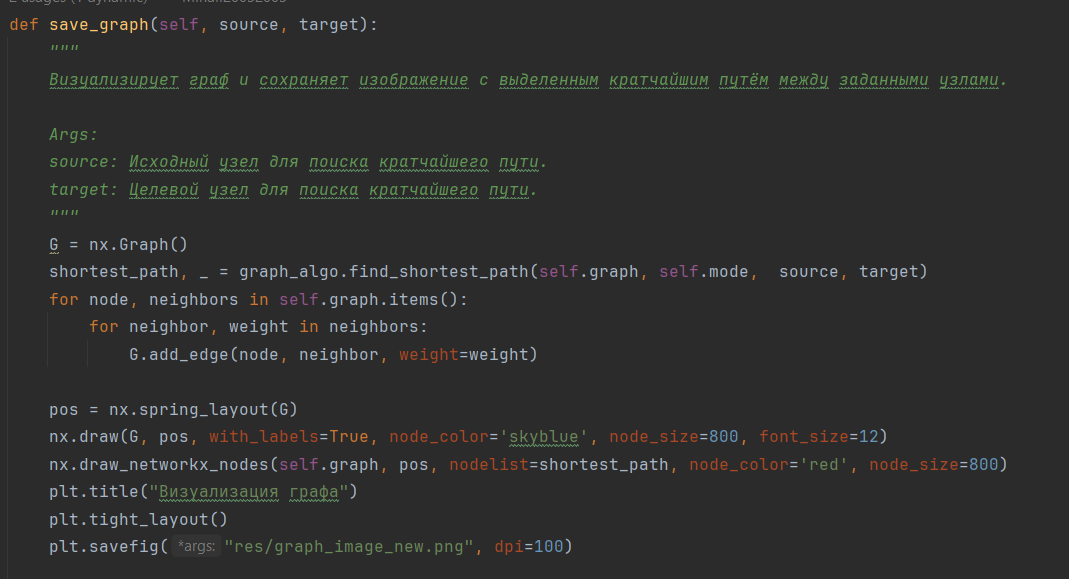


Рисунок 6. Метод def save\_graph

*Методы graph\_algo.py*

1. **def dijkstra(graph\_, source: int)** (рис. 7) –– выполняет алгоритм Дейкстры для нахождения кратчайших путей от заданного исходного узла до всех остальных узлов в графе.

**Args:**

**graph\_(dict)**: Словарь, представляющий граф, где ключи — это узлы, а значения - список смежных узлов с их весами.

**source (int)**: Исходный узел для поиска кратчайших путей.

**Returns:**

**dict:** Словарь, содержащий кратчайшие расстояния от исходного узла до всех остальных узлов в графе.

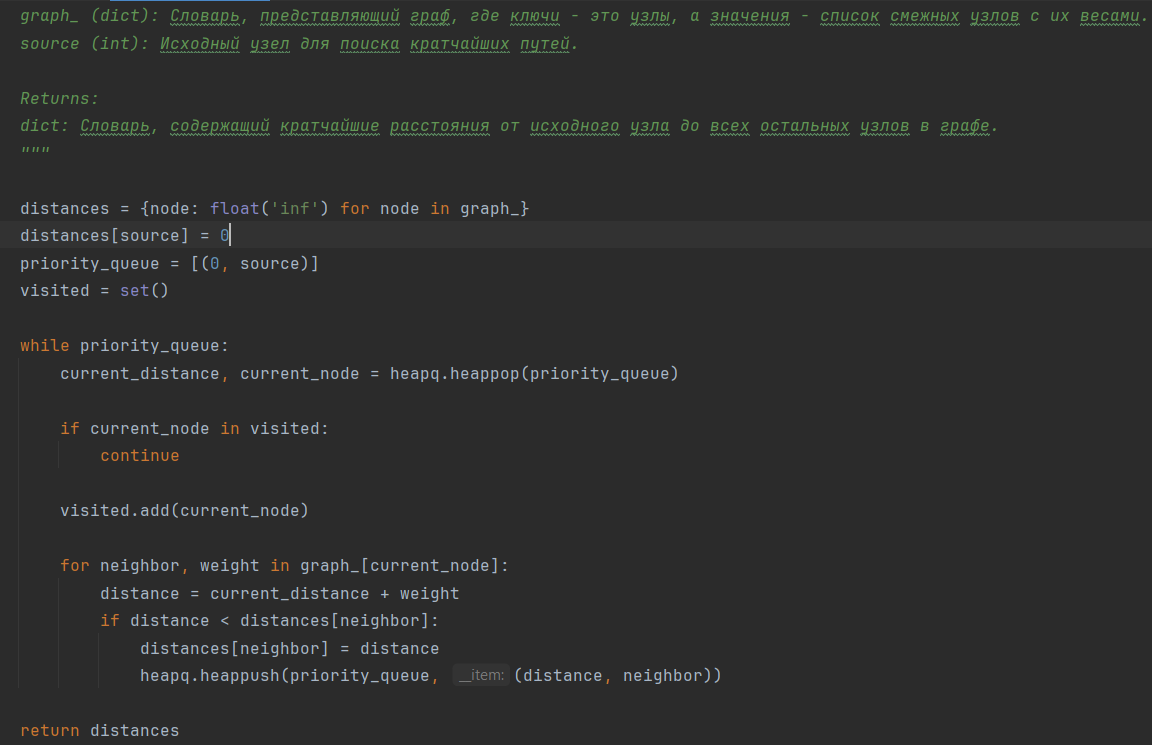


Рисунок 7. Метод def dijkstra.

1. **def bellman\_ford(graph\_, source: int)** (рис. 8)- Выполняет алгоритм Беллмана-Форда для нахождения кратчайших путей от заданного исходного узла до всех остальных узлов в графе.

**Args**:

**graph\_ (dict)**: Словарь, представляющий граф, где ключи — это узлы, а значения - список смежных узлов с их весами.

**source (int)**: Исходный узел для поиска кратчайших путей.

**Returns**:

**dict**: Словарь, содержащий кратчайшие расстояния от исходного узла до всех остальных узлов в графе

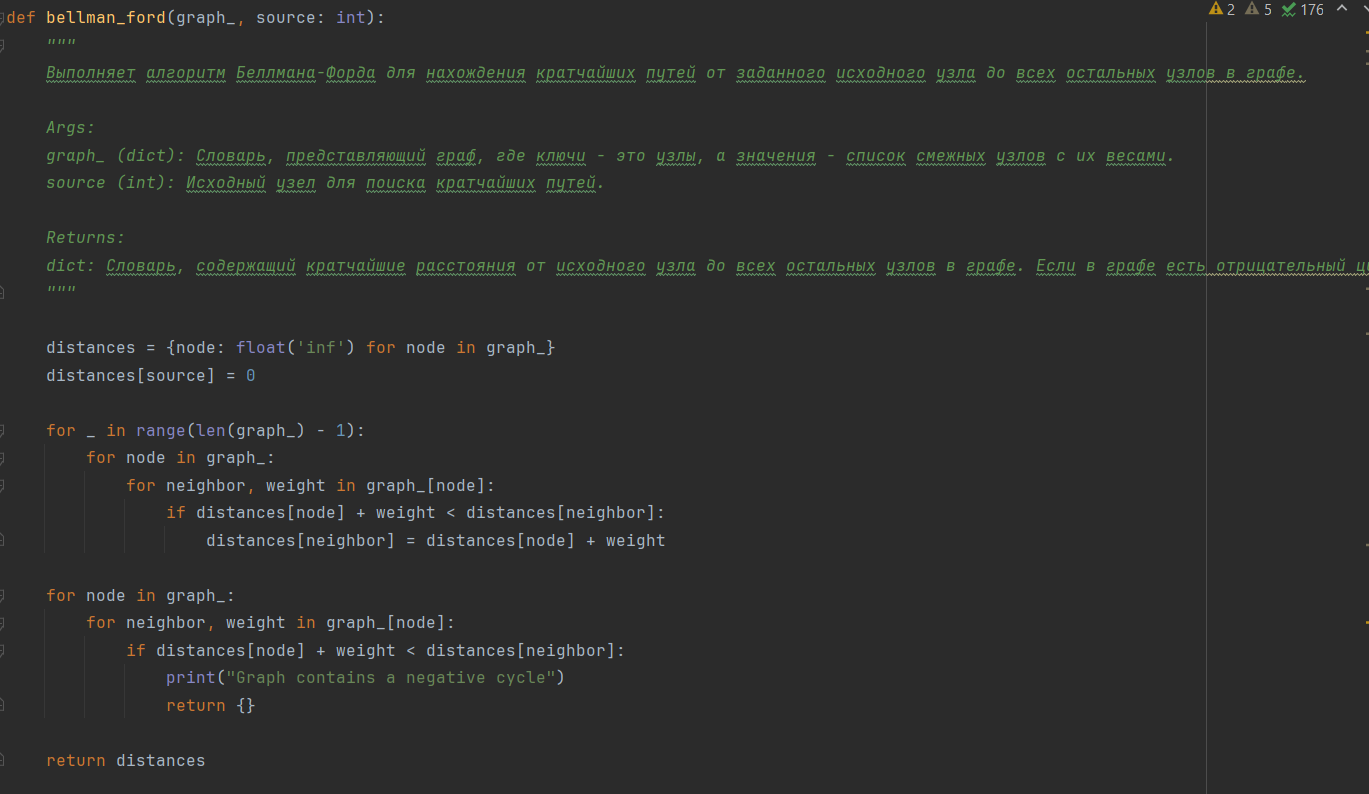


Рисунок 8. Метод def bellman\_ford

**Поиск кратчайшего пути (буде описан в дальнейшем).**

1. **def find\_shortest\_path(graph\_, mode, source: int, target: int)** (рис. 9)

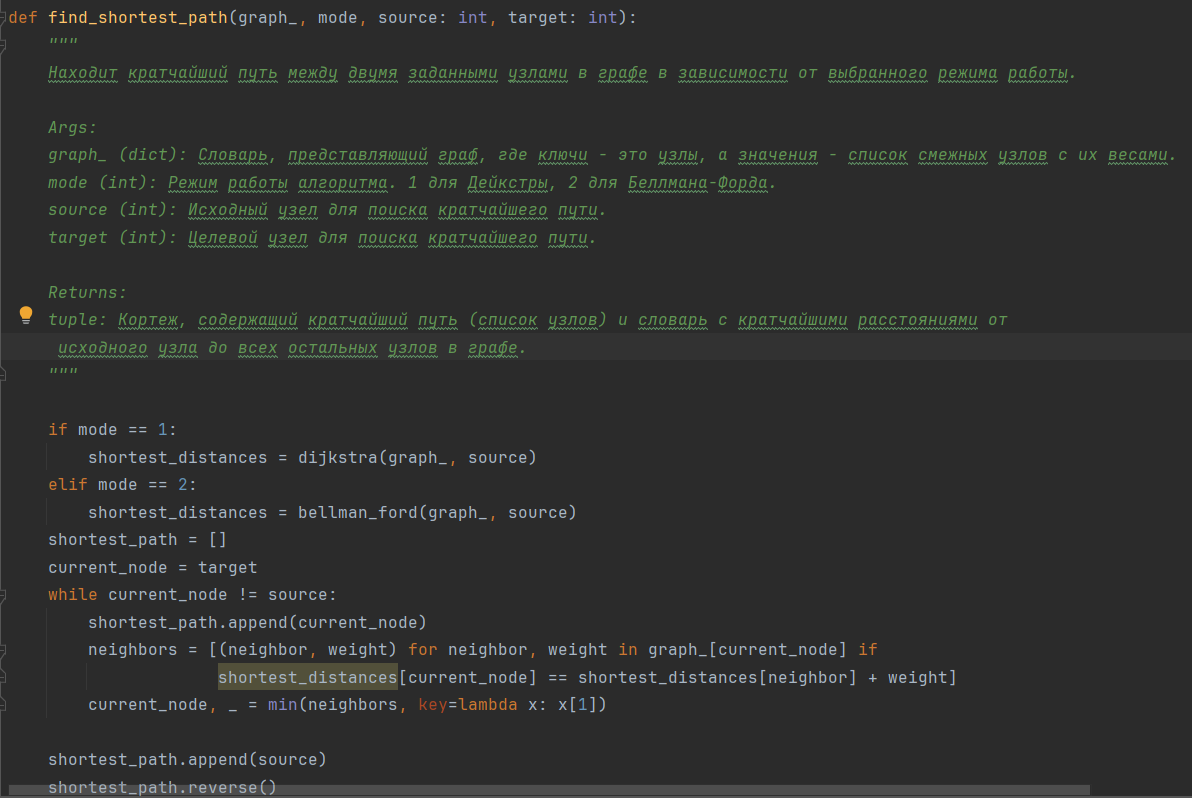


Рисунок 9. Метод find\_shortest\_path

1. **print\_shortes\_path()** (рис. 10) – вывод.

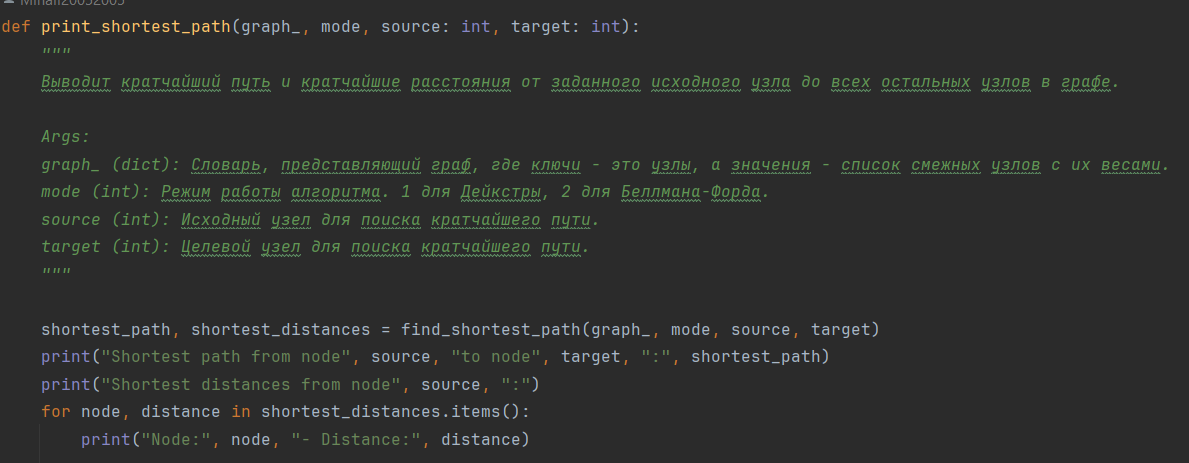


Рисунок 10. Метод print\_shortes\_path

*Описание классов*

**class Graph** – основной класс, предоставляющий функционал для работы с графом.

**class MainWindow** – главное окно, позволяющее генерировать граф, конфигурация которого определяется случайным образом.

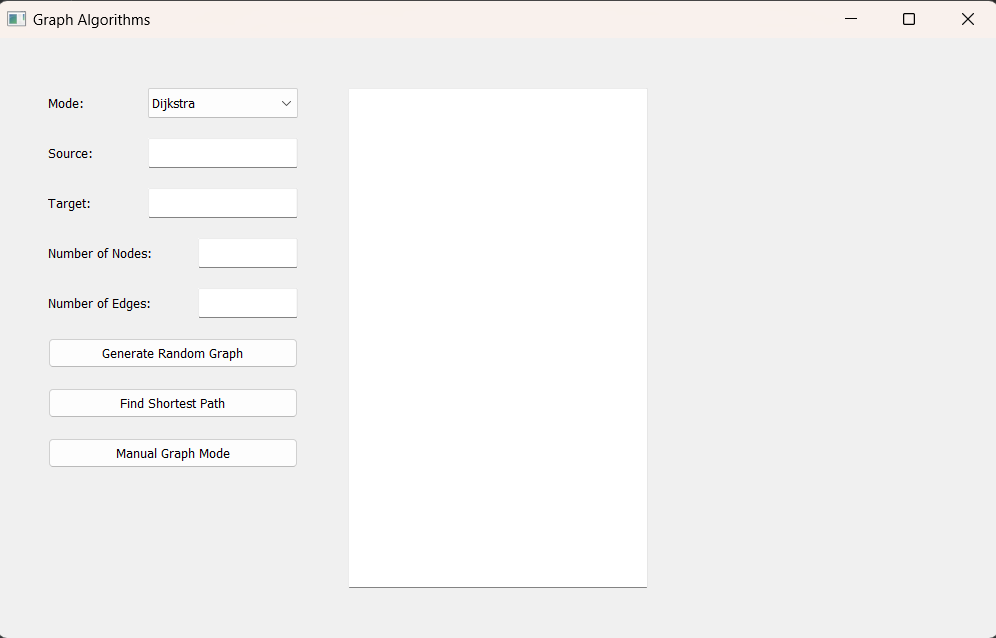


Рисунок 11. Сlass MainWindow

**class SecondWindow** – окно для ручной настройки конфигурации графа.

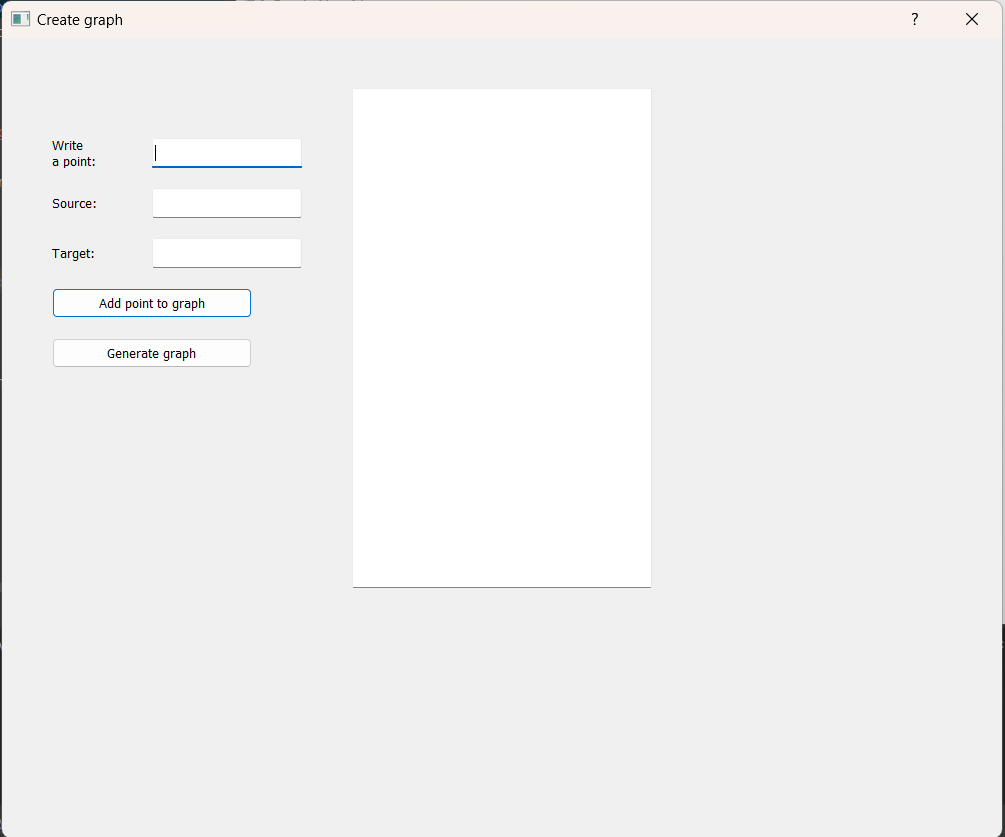


Рисунок 12. Сlass SecondWindow

В режиме генерации случайного графа пользователь вводит **начальную и конечную точку пути, количество вершин и ребер**. Нажимая кнопку **Generate Random Graph** пользователь генерирует граф, небольшое изображение которого отображается в правой части.

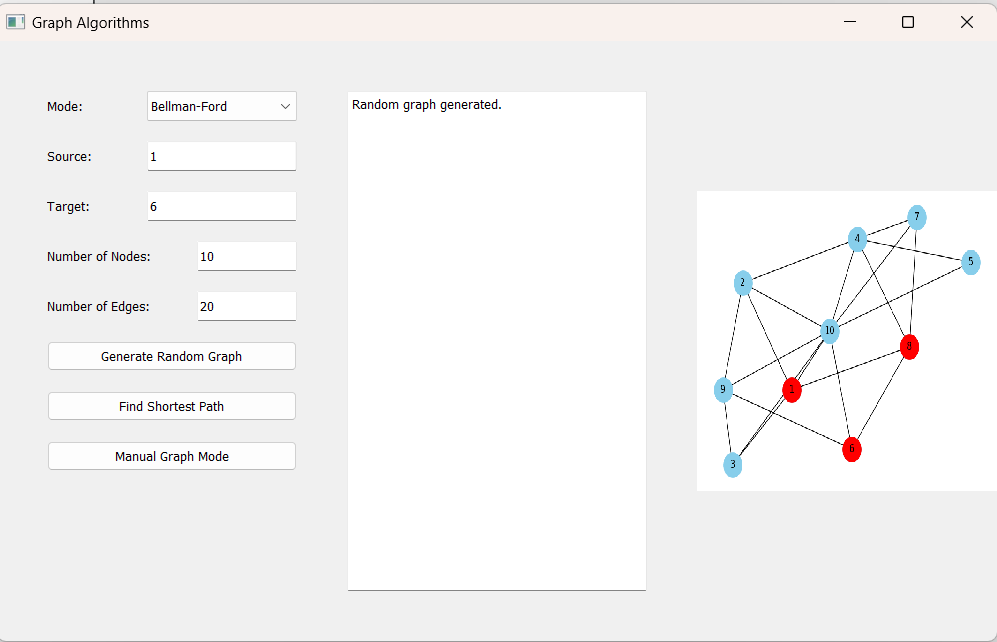


Рисунок 13. Генерация графа

Также данное изображение сохраняется в директорию res.

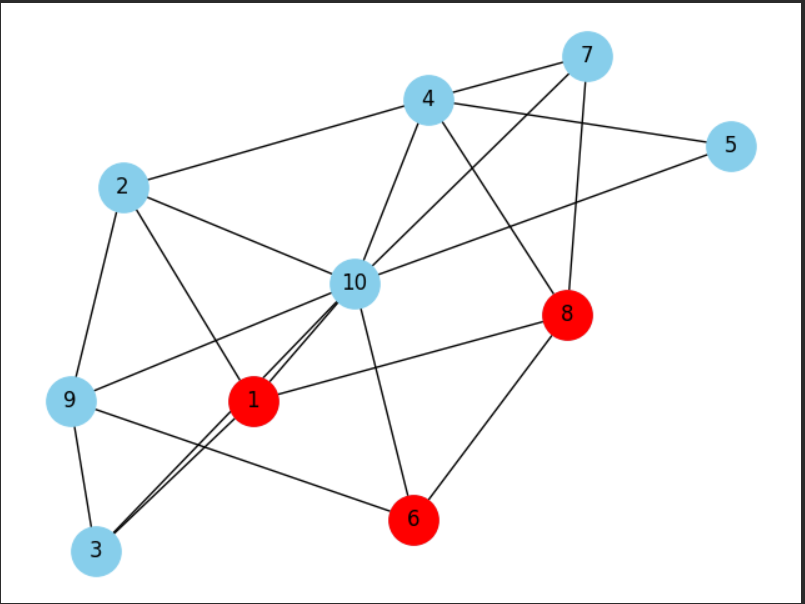


Рисунок 14. Визуализация работы алгоритма

При нажатии на кнопку **Find Shortest Path** выводится кратчайший путь, а также записываются расстояние от точки начала до всех других (обработка идет одним из двух алгоритмов).

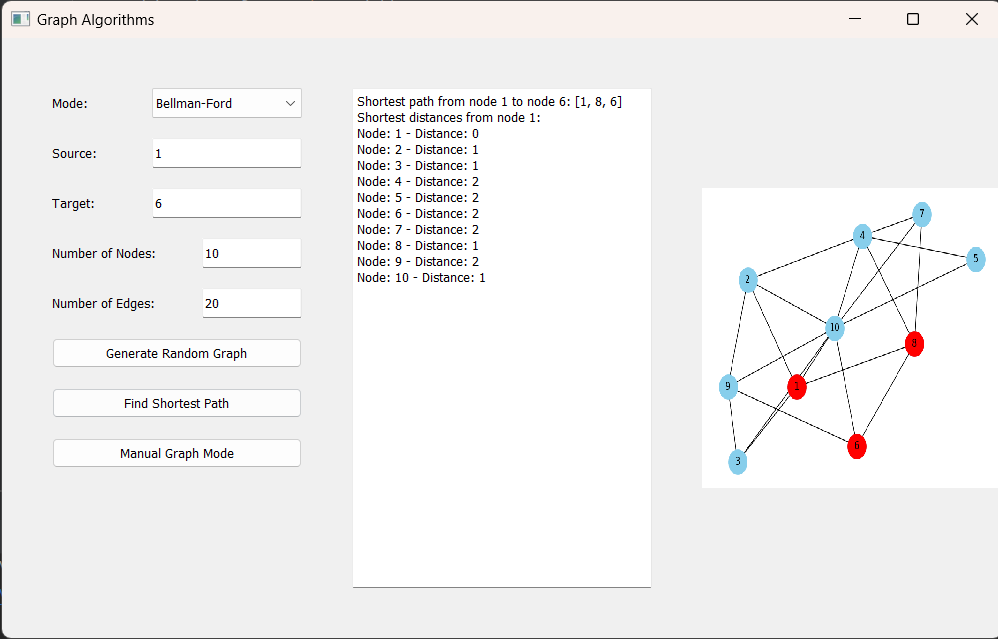


Рисунок 15. Вывод отработавшего алгоритма

При нажатии на кнопку **Manual Graph Mode** – программа переходит в ручной режим настройки. Необходимо ввести данные графа в формате

**begin end weight,**

где **begin** – точка начала дуги, **end** – точка конца дуги, **weight** – вес дуги

При нажатии на кнопку **Add point to graph** данная точка добавляется в граф, далее нажимаем кнопку **Generate graph** и наблюдаем сгенерированный граф с найденным кратчайшим путем (от **Source** до **Target**).

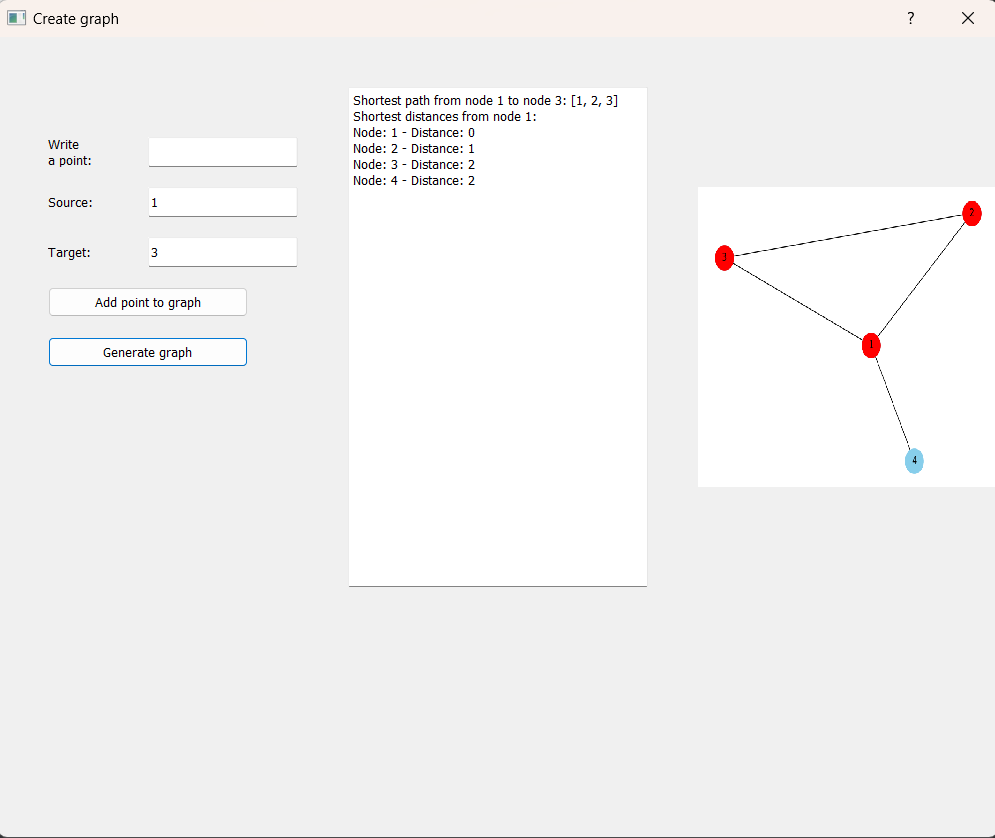


Рисунок 16. Пример рабочей отработавшей программы

*Для запуска программы сделайте следующие действия:*

1. Для запуска требуются библиотеки matplotlib, networkx, PyQt5.

Установите их следующими командами

**pip3 install [название модуля]**

1. Запустите файл main командой

python main.py

**Алгоритмы**

1. *Алгоритм Дийкстры*

*Релизация на Python*

1. **Инициализация переменных**:

* **distances**: это словарь, который будет содержать текущие кратчайшие расстояния от исходного узла до всех других узлов. Изначально все расстояния устанавливаются на бесконечность, кроме исходного узла, для которого расстояние устанавливается на 0.
* **priority\_queue**: это очередь с приоритетом, которая будет хранить пары (расстояние, узел). Она инициализируется кортежем (0, source), чтобы начать алгоритм с исходного узла.
* **visited**: это множество для отслеживания посещенных узлов.

1. **Извлечение узла из очереди**: на каждой итерации извлекается узел с наименьшим текущим расстоянием из priority\_queue.
2. **Проверка посещаемости узла**: если узел уже был посещен (то есть он находится в visited), мы пропускаем его и переходим к следующей итерации.
3. **Просмотр соседей текущего узла**: для каждого соседа текущего узла вычисляется новое расстояние от исходного узла через текущий узел. Если это расстояние меньше, чем текущее расстояние до соседа, мы обновляем его в словаре distances и добавляем пару (новое расстояние, сосед) в priority\_queue.
4. **Добавление узла в посещенные**: после того как мы просмотрели всех соседей текущего узла, мы добавляем текущий узел в множество visited, чтобы не рассматривать его снова.
5. **Возвращение результата**: По завершении цикла возвращается словарь distances, содержащий кратчайшие расстояния от исходного узла до всех других узлов в графе.
6. *Алгоритм Беллмана – Форда*
7. **Инициализация переменных**:
8. **distances**: это словарь, который будет содержать текущие кратчайшие расстояния от исходного узла до всех других узлов. Изначально все расстояния устанавливаются на бесконечность, кроме расстояния до исходного узла, которое устанавливается на 0.
9. **Циклы для релаксации ребер**:
10. Алгоритм выполняет **len(graph\_) - 1** итераций, где **graph\_** — это словарь, представляющий граф. Это количество итераций достаточно для обновления всех возможных кратчайших путей от исходного узла ко всем другим узлам в графе.
11. На каждой итерации внешнего цикла происходит проход по всем узлам графа (**for node in graph\_**), а затем внутренний цикл просматривает всех соседей каждого узла и рассматривает возможность улучшения текущих расстояний до соседних узлов.
12. **Обновление расстояний**:
13. Если расстояние от исходного узла до текущего узла плюс вес ребра до соседнего узла меньше текущего кратчайшего расстояния до соседнего узла, расстояние до соседнего узла обновляется на новое значение.
14. **Проверка наличия отрицательного цикла**:
15. После завершения основных итераций алгоритма происходит еще один проход по всем ребрам графа для проверки наличия отрицательного цикла.
16. Если на этом этапе обнаруживается, что какое-то расстояние может быть еще улучшено, это означает наличие отрицательного цикла в графе.
17. **Возвращение результата**:
18. По завершении алгоритма возвращается словарь **distances**, содержащий кратчайшие расстояния от исходного узла до всех других узлов в графе. Если в графе обнаружен отрицательный цикл, возвращается пустой словарь.

**Список литературы**

1. Актуальность и применение – Задача кратчайшего пути [Электронный ресурс] – Режим доступа:

<https://vuzlit.com/829896/aktualnost_primenenie?ysclid=lvusorcus1365416429>

1. Алгоритм Дейкстры – что это такое [Электронный ресурс] – Режим доступа:

[https://evmservice.ru/blog/algoritm-dejkstry-chto-eto-takoe/](https://evmservice.ru/blog/algoritm-dejkstry-chto-eto-takoe/#%D0%9A%D0%B5%D0%BC_%D0%B8_%D0%BA%D0%BE%D0%B3%D0%B4%D0%B0_%D0%B1%D1%8B%D0%BB_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%BD)

1. Алгоритм Беллмана-Форда [Электронный ресурс] – Режим доступа:  
   <https://kvodo.ru/bellman-ford-algorithm.html?ysclid=lvv5emtjei512767601>
2. Said Sryheni, Dijkstra’s vs Bellman-Ford Algorithm [Электронный ресурс] – Режим доступа:  
   <https://translated.turbopages.org/proxy_u/en-ru.ru.cfc5fda1-6638ffb9-f66d2389-74722d776562/https/www.baeldung.com/cs/dijkstra-vs-bellman-ford>
3. Теория графов. Термины и определения [Электронный ресурс] – Режим доступа:  
   <https://habr.com/ru/companies/otus/articles/568026/>

# Тренировки по алгоритмам 4.0. Лекция 3: Кратчайшие пути во взвешенных графах [Электронный ресурс] – Режим доступа:

# <https://www.youtube.com/watch?v=sGU4xxp9N3o>

1. Толстиков А., Куликов А., Принципы построения алгоритмов [Электронный ресурс] – Режим доступа:   
   [https://education.yandex.ru/handbook/algorithms/article/principy-postroeniya-algoritmov](https://vk.com/away.php?utf=1&to=https%3A%2F%2Feducation.yandex.ru%2Fhandbook%2Falgorithms%2Farticle%2Fprincipy-postroeniya-algoritmov)
2. Горденко М., Задача поиска кратчайшего пути в графе [Электронный ресурс] – Режим доступа:  
   <https://education.yandex.ru/handbook/algorithms/article/zadacha-poiska-kratchajshego-puti-v-grafe>