**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Решение задачи о коммивояжере с помощью алгоритма имитации отжига»**

**Студент гр. 23Б16-пу**

**Горынцев Р.Н.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [**Цель работы** 3](#_Toc194712835)
2. [**Описание задачи (формализация задачи)** 3](#_Toc194712836)
3. [**Теоретическая часть** 4](#_Toc194712837)
4. [**Основные шаги программы** 5](#_Toc194712838)
5. [**Блок схема программы** 7](#_Toc194712839)
6. [**Описание программы** 8](#_Toc194712840)
7. [**Рекомендации пользователя** 10](#_Toc194712841)
8. [**Контрольный пример** 11](#_Toc194712842)
9. [**Анализ результатов работы** 14](#_Toc194712843)
10. [**Сравнение результатов алгоритма имитации отжига с методом ближайших соседей** 17](#_Toc194712844)
11. [**Вывод** 20](#_Toc194712845)
12. [**Листинг** 21](#_Toc194712846)

# **Цель работы**

Целью данной работы является исследование особенностей решения задачи о коммивояжере с помощью алгоритма имитации отжига. Задача заключается в нахождении кратчайшего гамильтонова цикла в графе, который проходит через все вершины графа ровно один раз и возвращается в исходную вершину.

# **Описание задачи (формализация задачи)**

Задача о коммивояжере заключается в нахождении кратчайшего цикла, проходящего через все вершины графа ровно один раз. Граф представлен в виде взвешенного орграфа, где рёбра имеют определённые веса (расстояния между вершинами). Формализация задачи включает:

1. **Изучение задачи и алгоритма**Разобраться с условиями задачи коммивояжера и понять принцип работы алгоритма имитации отжига. Рассмотреть стандартную версию алгоритма и вариант с модификацией.
2. **Реализация программы**Написать программу, которая решает задачу двумя способами: стандартным алгоритмом имитации отжига и его модифицированной версией.
3. **Сравнение алгоритмов**Сравнить работу обычного и измененного алгоритмов. Фиксировать время выполнения и качество получаемых решений. Сравнить с методом ближайших соседей.
4. **Анализ эффективности**Изучить, как модификация влияет на точность и скорость работы алгоритма.

# **Теоретическая часть**

Метод имитации отжига — это эвристический алгоритм для решения задачи коммивояжёра, вдохновлённый процессом медленного охлаждения металла. Основная идея заключается в том, чтобы на каждом шаге случайно изменять текущий маршрут и иногда допускать ухудшения, чтобы избежать застревания в локальных минимумах. Алгоритм работает следующим образом:

1. **Инициализация:** Выбираем начальный маршрут и задаём начальную температуру.
2. **Генерация нового решения:** Cлегка изменяем текущий маршрут (например, переставляя два города)**.**
3. **Принятие решения:**
   * Если новый маршрут короче, всегда принимаем его.
   * Если он длиннее, иногда принимаем его (вероятность зависит от температуры).
4. **Охлаждение:** Постепенно уменьшаем температуру, снижая вероятность принятия плохих решений.
5. **Завершение**: Когда температура становится очень низкой, возвращаем лучший найденный маршрут.

**Преимущества**:

* Может избегать локальных минимумов за счёт временного ухудшения решений.
* Гибкий и применим к разным задачам оптимизации.
* При правильной настройке даёт хорошие результаты.

**Недостатки**:

* Требует подбора параметров (начальная температура, скорость охлаждения).
* Не гарантирует нахождение оптимального решения.

# **Основные шаги программы**

**1. Инициализация графа и интерфейса**

* Создаётся пустой граф с использованием библиотеки networkx.
* Инициализируется графический интерфейс с помощью библиотеки tkinter.
* Настраиваются кнопки, таблица для отображения рёбер и холст для визуализации графа.
* Вводятся входные данные.

**2. Добавление вершин**

* Пользователь кликает на холст, чтобы добавить вершину.
* Координаты клика сохраняются, и вершина добавляется в граф с уникальным ID.
* Вершина отображается на холсте в виде круга с текстом, указывающим её ID.

**3. Добавление рёбер**

* Пользователь выбирает две вершины, кликая на них поочерёдно.
* После выбора двух вершин программа запрашивает вес ребра.
* Ребро добавляется в граф с указанным весом и отображается на холсте в виде линии со стрелкой.

**4. Реализация алгоритма имитации отжига**

1. **Подготовка:**
   * Создаём начальный маршрут
   * Устанавливаем начальную температуру и скорость её снижения.
2. **Главный процесс (повторяем, пока система не остынет):**
   * **Пробуем новый вариант:**
     + Немного изменяем текущий маршрут.
   * **Проверяем результат:**
     + Сравниваем длину нового маршрута с текущим.
   * **Решаем, принять ли изменения:**
     + Если маршрут стал короче — берём его.
     + Если стал длиннее — берём с какой-то вероятностью
   * **Понижаем температуру:**
     + Постепенно уменьшаем вероятность принимать плохие решения.
   * **Запоминаем лучший вариант:**
     + Сохраняем самый короткий найденный маршрут.
3. **Завершение:**
   * Возвращаем лучший маршрут из всех найденных.

**5. Расчёт длины пути**

* Программа вычисляет суммарный вес рёбер, входящих в найденный цикл.
* Длина пути отображается в интерфейсе.

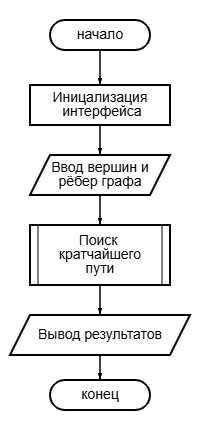
**6. Визуализация графа и пути**

* Граф отображается на холсте:
  + Вершины отображаются в виде кругов с текстом, указывающим их ID.
  + Рёбра отображаются в виде линий со стрелками.
* Найденный путь выделяется красным цветом.

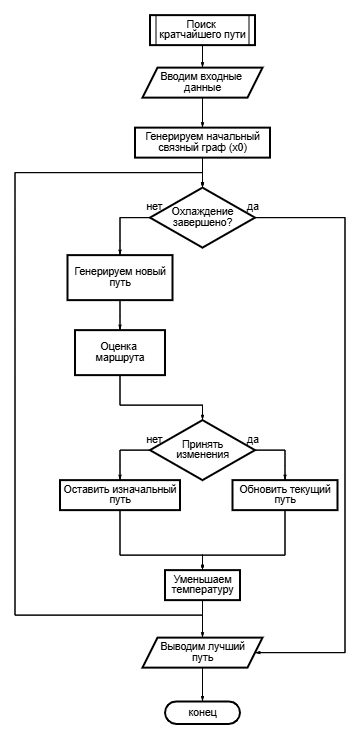
**7. Очистка графа**

* Пользователь может очистить граф, нажав кнопку "Очистить".
* Все вершины и рёбра удаляются, и интерфейс сбрасывается в исходное состояние.

# **Блок схема программы**



*Рис 1: Блок схема программы*





*Рис 2: Блок схема подпрограммы*

# **Описание программы**

Программная реализация выполнена на языке Python 3.12 с использованием библиотек tkinter, network, random и math. Программа организована в виде единого файла, основная цель которого — решение задачи о коммивояжере с применением метода имитации отжига. В рамках программы реализован класс GraphBuilder, который объединяет всю логику работы с графом и графическим интерфейсом. В процессе разработки программы использовались функции, каждая из которых имеет чётко определённое назначение:

*Таблица 1. Описание функций*

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| \_\_init\_\_ | Инициализирует графический интерфейс и создаёт пустой граф. | None |
| create\_connected\_graph | Создаёт связный граф с заданным количеством вершин | None |
| on\_canvas\_click | Обрабатывает клик мыши на холсте для добавления вершин и рёбер. | None |
| on\_canvas\_right\_click | Обрабатывает правый клик мыши для удаления или изменения рёбер. | None |
| show\_edge\_context\_menu | Отображает контекстное меню для редактирования или удаления ребра. | None |
| edit\_edge\_length | Изменяет вес ребра. | None |
| find\_node | Находит вершину по координатам клика мыши. | None |
| find\_edge | Находит ребро по координатам клика мыши. | None |
| add\_edge | Добавляет ребро между двумя вершинами с указанным весом. | None |
| remove\_edge | Удаляет ребро из графа. | None |
| calculate\_path | Вычисляет кратчайший путь с использованием метода ближайшего соседа. | None |
| has\_hamiltonian\_cycle | Проверяет, существует ли в графе гамильтонов цикл | bool |
| calculate\_path\_with\_modification | Вычисляет кратчайший путь с модификацией | None |
| simulated\_annealing | Реализует стандартный алгоритм имитации отжига. | tuple (путь, длина) |
| simulated\_annealing\_cauchy | Реализует модифицированный алгоритм имитации отжига с критерием Коши. | tuple (путь, длина) |
| calculate\_path\_distance | Вычисляет длину пути на основе найденного цикла. | float |
| update\_edges\_table | Обновляет таблицу рёбер в интерфейсе. | None |
| draw\_graph | Визуализирует граф на холсте. | None |
| draw\_original\_graph | Визуализирует граф с выделенным кратчайшим путём. | None |
| clear\_all | Очищает граф и интерфейс, сбрасывая все данные. | None |

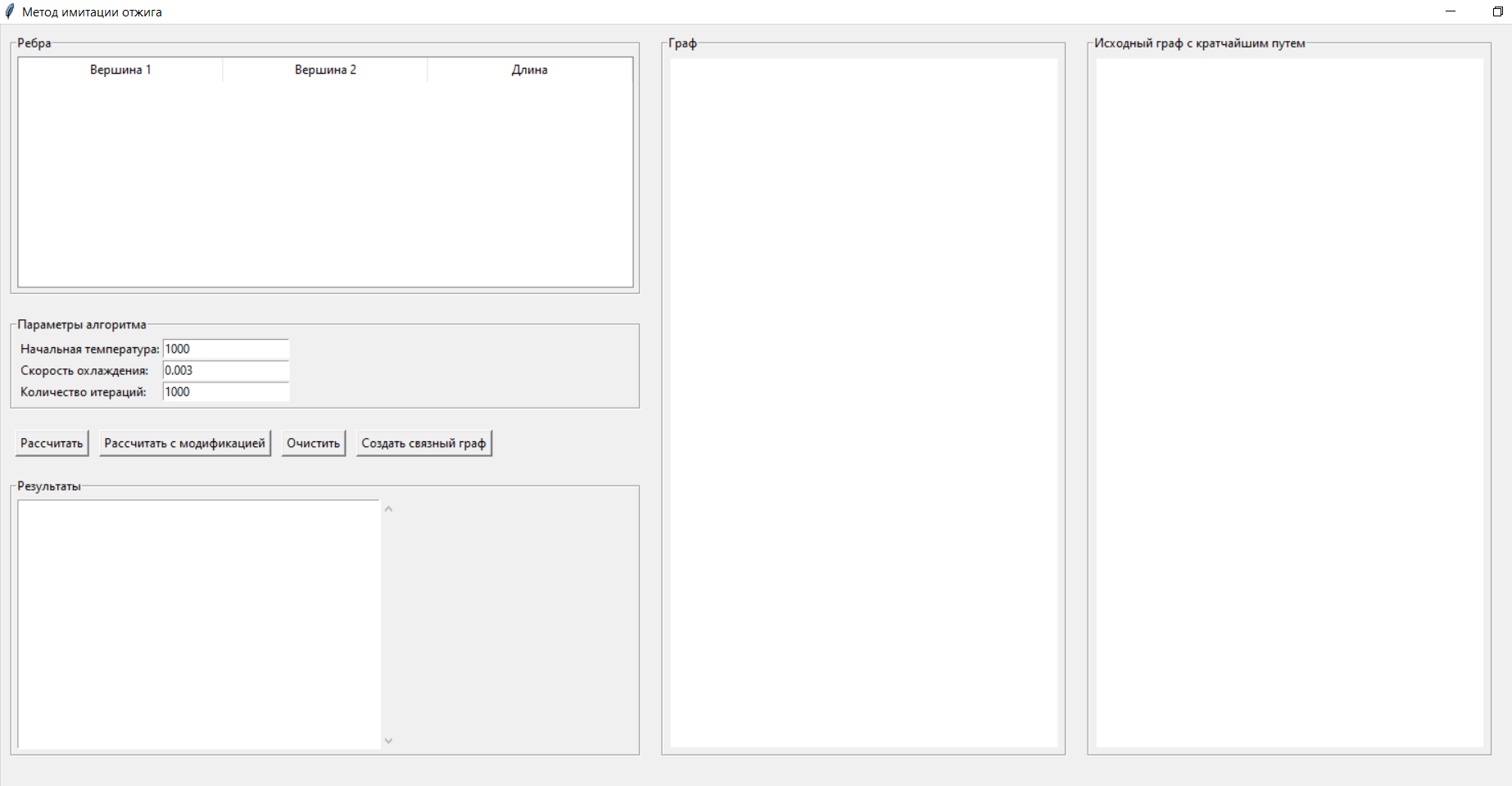
# **Рекомендации пользователя**

1. Убедитесь, что у вас установлен Python 3.12 и библиотеки tkinter, networkx, math, random.
2. Запустите программу.
3. Введите входные данные для алгоритма.
4. Добавление вершин.
   * Кликните левой кнопкой мыши на холст, чтобы добавить вершину. Каждая вершина будет автоматически пронумерована.
5. Добавление рёбер.
   * Кликните левой кнопкой мыши на две вершины поочерёдно, чтобы добавить ребро. Введите вес ребра.
6. Редактирование и удаление рёбер.
   * Кликните правой кнопкой мыши на ребро, чтобы открыть контекстное меню. Выберите "Изменить длину", чтобы изменить вес ребра, или "Удалить ребро", чтобы удалить его.
7. Расчёт кратчайшего пути.
   * Нажмите кнопку "Рассчитать", чтобы найти кратчайший путь с использованием имитации отжига.
   * Для улучшения результата используйте кнопку "Рассчитать с модификацией", которая запускает модификацию Коши.
8. Очистка графа.
   * Нажмите кнопку "Очистить", чтобы удалить все вершины и рёбра и начать работу с новым графом.

# **Контрольный пример**

**1. Запуск программы:**

* Пользователь запускает программу. На экране появляется интерфейс с полем для ввода данных о графе, кнопками для выполнения расчётов и областями для вывода результатов (Рис. 3).



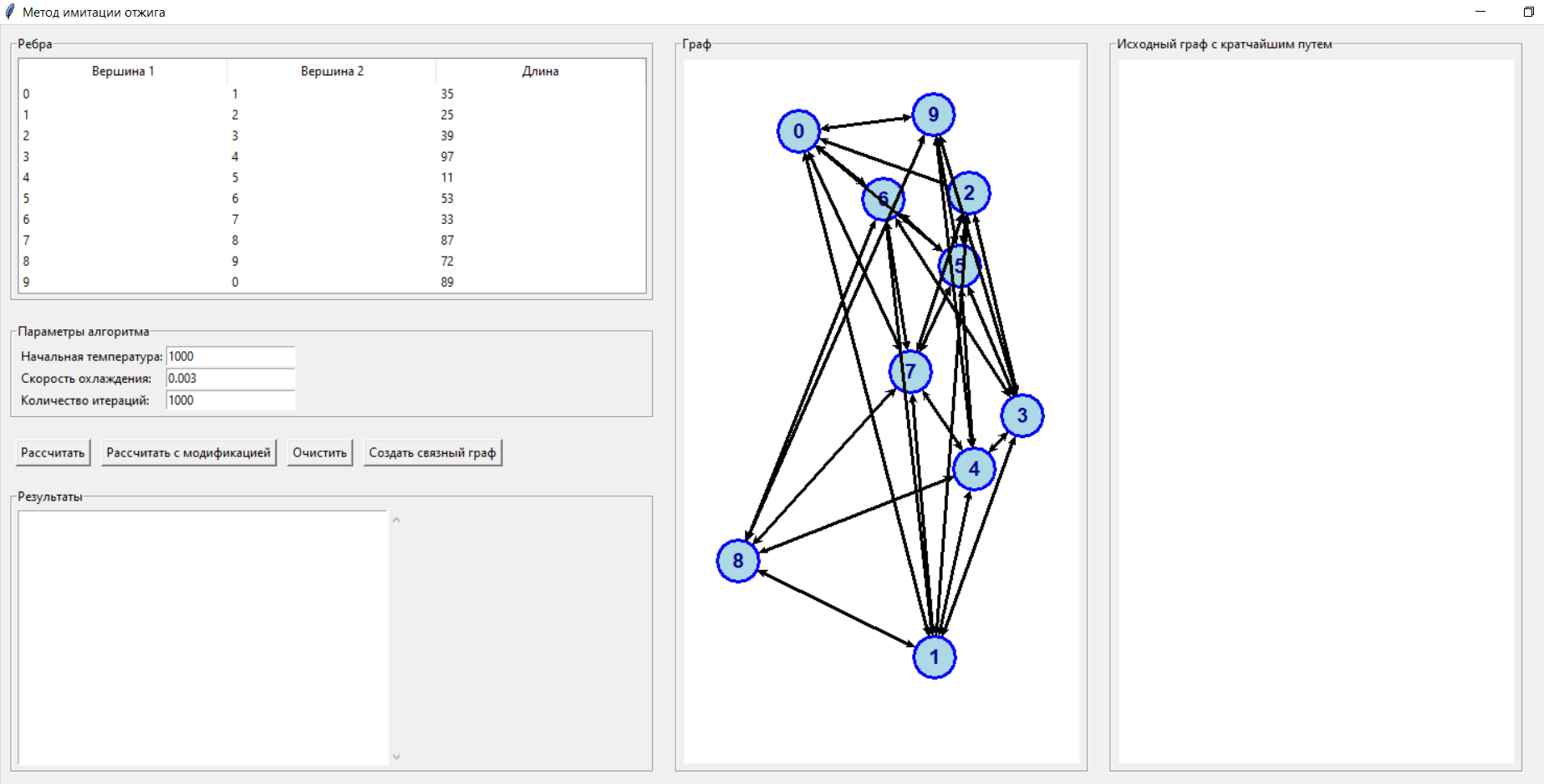
*Рис 3: Интерфейс программы*

**2. Добавление вершин:**

* + Пользователь кликает левой кнопкой мыши на холсте для добавления вершины
  + Программа автоматически создаёт вершину с номером (начиная с 0) и отображает её на холсте.
  + Пользователь повторяет этот шаг для добавления всех вершин.

**3. Добавление вершин:**

* 1. Пользователь выбирает первую вершину ребра, кликая левой кнопкой мыши на ней (например, вершина 0).
  2. Затем пользователь кликает левой кнопкой мыши на второй вершине ребра (например, вершина 1).
  3. Программа запрашивает длину ребра между выбранными вершинами.
  4. Пользователь вводит длину ребра и нажимает "ОК".
  5. Программа добавляет ребро между вершинами с длиной и отображает его на холсте.
  6. Пользователь повторяет эти шаги для добавления всех рёбер.

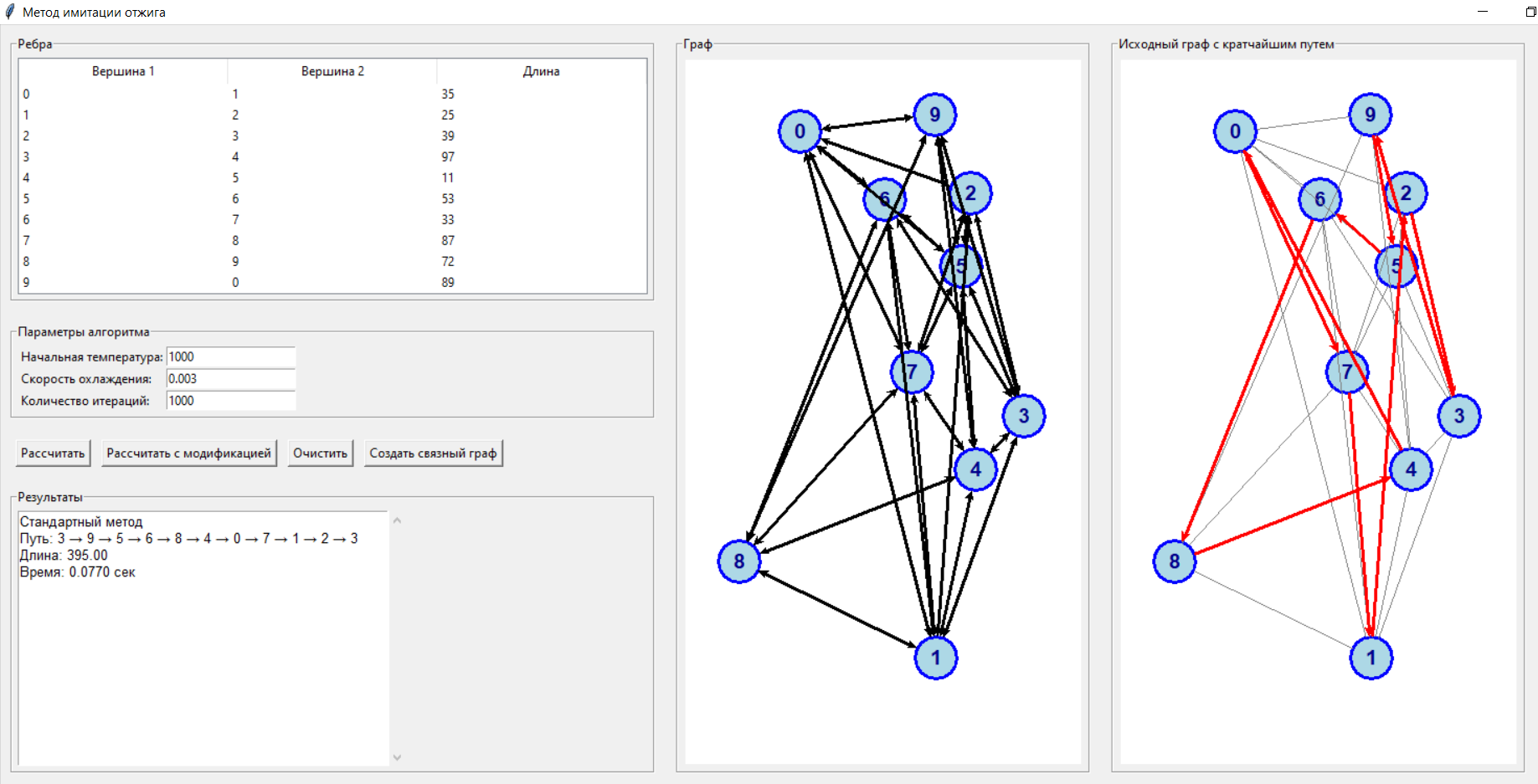


*Рис 4: пример окна программы*

**4. Расчёт и результаты:**

После ввода данных пользователь выполняет расчёты и получает результаты:

1. Обычный метод имитации отжига:
   * Пользователь нажимает кнопку "Рассчитать".
2. Метод имитации отжига с модификацией:
   * Пользователь нажимает кнопку "Рассчитать с модификацией".
3. Визуализация графа:
   * Программа отображает исходный граф с выделенным кратчайшим путём.
   * На Рис.5 видно, как вершины соединены рёбрами, а найденный путь выделен красным цветом.



*Рис 5: пример результатов программы*

# **Анализ результатов работы**

Имитация отжига — это вероятностный алгоритм, который имитирует процесс охлаждения металла. Он позволяет избегать застревания в локальных минимумах за счёт случайных переходов, вероятность которых уменьшается со временем.

Основные параметры алгоритма

* **T (температура)** – контролирует вероятность принятия худших решений.
* **α (alpha)** – определяет, как быстро снижается температура.
* **Количество итераций**

Для графа с 10 вершинами использовались следующие параметры:

* 1 итерация: T = 50, alpha = 0,85, количество итераций = 50
* 2 итерация: T = 100, alpha = 0,9, количество итераций = 100
* 3 итерация: T = 50, alpha = 0,95, количество итераций = 100
* 4 итерация: T = 100, alpha = 0,99, количество итераций = 50
* 5 итерация: T = 150, alpha = 0,9, количество итераций = 200

*Таблица 2. Результаты графа с 10 вершинами*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Время выполнения (без модификации) | Время выполнения (с модификацией) | Кратчайшая длина пути (без модификацией) | Кратчайшая длина пути (с модификацией) |
| 1 | 0.0070 | 0.0060 | 300 | 268 |
| 2 | 0.0100 | 0.0090 | 280 | 287 |
| 3 | 0.0090 | 0.0090 | 265 | 281 |
| 4 | 0.0061 | 0.0060 | 284 | 242 |
| 5 | 0.0140 | 0.0101 | 297 | 286 |

Для графа с 20 вершинами использовались следующие параметры:

* 1 итерация: T = 100, alpha = 0,85, количество итераций = 100
* 2 итерация: T = 200, alpha = 0,9, количество итераций = 200
* 3 итерация: T = 100, alpha = 0,95, количество итераций = 200
* 4 итерация: T = 200, alpha = 0,99, количество итераций = 100
* 5 итерация: T = 300, alpha = 0,9, количество итераций = 300

*Таблица 3. Результаты графа с 20 вершинами*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Время выполнения (без модификации) | Время выполнения (с модификацией) | Кратчайшая длина пути (без модификации) | Кратчайшая длина пути (с модификацией) |
| 1 | 0.0321 | 0.0210 | 983 | 911 |
| 2 | 0.0431 | 0.0401 | 886 | 882 |
| 3 | 0.0450 | 0.0521 | 822 | 894 |
| 4 | 0.0290 | 0.0280 | 839 | 953 |
| 5 | 0.0470 | 0.0581 | 821 | 902 |

Для графа с 30 вершинами использовались следующие параметры:

* 1 итерация: T = 200, alpha = 0,85, количество итераций = 200
* 2 итерация: T = 300, alpha = 0,9, количество итераций = 300
* 3 итерация: T = 200, alpha = 0,95, количество итераций = 300
* 4 итерация: T = 400, alpha = 0,99, количество итераций = 200
* 5 итерация: T = 500, alpha = 0,9, количество итераций = 500

*Таблица 4. Результаты графа с 30 вершинами*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Время выполнения (без модификации) | Время выполнения (с модификацией) | Кратчайшая длина пути (без модификации) | Кратчайшая длина пути (с модификацией) |
| 1 | 0.0170 | 0.0980 | 1809 | 1533 |
| 2 | 0.0180 | 0.1381 | 2056 | 1466 |
| 3 | 0.0140 | 0.1421 | 1979 | 1366 |
| 4 | 0.0130 | 0.0881 | 1973 | 1441 |
| 5 | 0.0200 | 0.1921 | 2048 | 1334 |

**Влияние параметров**

* + **Температура (T)**: Чем выше начальная температура, тем лучше алгоритм исследует пространство решений, но увеличивается время работы. Например, при T=150 для 10 вершин длина пути уменьшилась до 286.
  + **Скорость охлаждения (alpha)**: Близкие к 1 значения (0.95-0.99) дают более точные результаты, но требуют больше времени. При alpha=0.99 для 10 вершин получен путь 242.
  + **Итерации**: Для 10 вершин достаточно 50-100 итераций, для 30 вершин требуется 300-500. Увеличение итераций улучшает результат, но после определенного предела эффективность падает.

**Сравнение алгоритмов**

* + **Модифицированный алгоритм**:

Находит более короткие пути (например, 242 против 284 для 10 вершин). На больших графах (30 вершин) время выполнения возрастает в 5-10 раз (0.1921с против 0.0200с).

* + **Стандартный алгоритм**:

Работает стабильно быстро (0.006-0.047с для 20 вершин).Часто застревает в локальных минимумах (путь 2048 против 1334 с модификацией для 30 вершин).

# **Сравнение результатов алгоритма имитации отжига с методом ближайших соседей**

Вспомним результаты метода ближайших соседей:

*Таблица 5. Результаты графа с 10 вершинами*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Время выполнения (без модификации) | Время выполнения (с модификацией) | Кратчайшая длина пути (без модификацией) | Кратчайшая длина пути (с модификацией) |
| 1 | 0.000000 | 0.000000 | 293 | 293 |
| 2 | 0.000000 | 0.001003 | 293 | 293 |
| 3 | 0.000997 | 0.000000 | 293 | 293 |
| 4 | 0.000000 | 0.000000 | 293 | 293 |
| 5 | 0.000000 | 0.001980 | 293 | 293 |

*Таблица 6. Результаты графа с 20 вершинами*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Время выполнения (без модификации) | Время выполнения (с модификацией) | Кратчайшая длина пути (без модификации) | Кратчайшая длина пути (с модификацией) |
| 1 | 0.000999 | 0.004018 | 714 | 344 |
| 2 | 0.000667 | 0.000305 | 714 | 344 |
| 3 | 0.000000 | 0.002033 | 714 | 344 |
| 4 | 0.000000 | 0.001011 | 714 | 344 |
| 5 | 0.000000 | 0.000000 | 714 | 344 |

*Таблица 7. Результаты графа с 30 вершинами*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Время выполнения (без модификации) | Время выполнения (с модификацией) | Кратчайшая длина пути (без модификации) | Кратчайшая длина пути (с модификацией) |
| 1 | 0.000998 | 0.003999 | 1084 | 459 |
| 2 | 0.000000 | 0.008995 | 1084 | 459 |
| 3 | 0.000000 | 0.007997 | 1084 | 459 |
| 4 | 0.000000 | 0.008037 | 1084 | 459 |
| 5 | 0.000976 | 0.008998 | 1084 | 459 |

1. Скорость работы

* Ближайший сосед — работает мгновенно даже на больших графах.
* Имитация отжига — требует больше времени, особенно на больших графах.

2. Качество решения

* На малых графах имитация отжига иногда находит более короткие пути, но разница незначительна.
* На средних и больших графах (20+ вершин) ближайший сосед с модификацией чаще дает лучшие результаты, причем с большим отрывом.

3. Стабильность

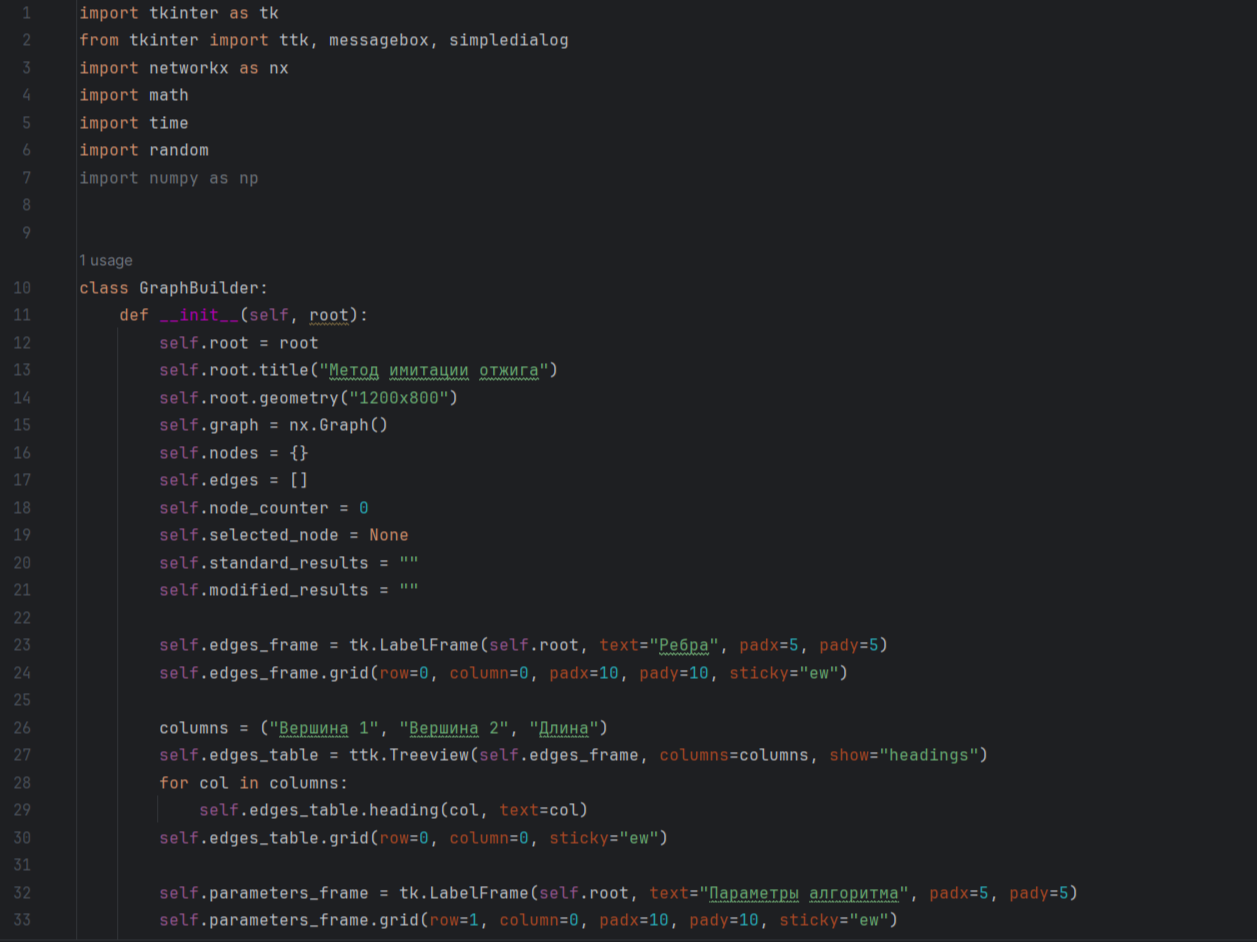
* Ближайший сосед всегда выдает один и тот же результат при одинаковых входных данных.
* Имитация отжига из-за случайности может давать разные решения при повторных запусках.

# **Вывод**

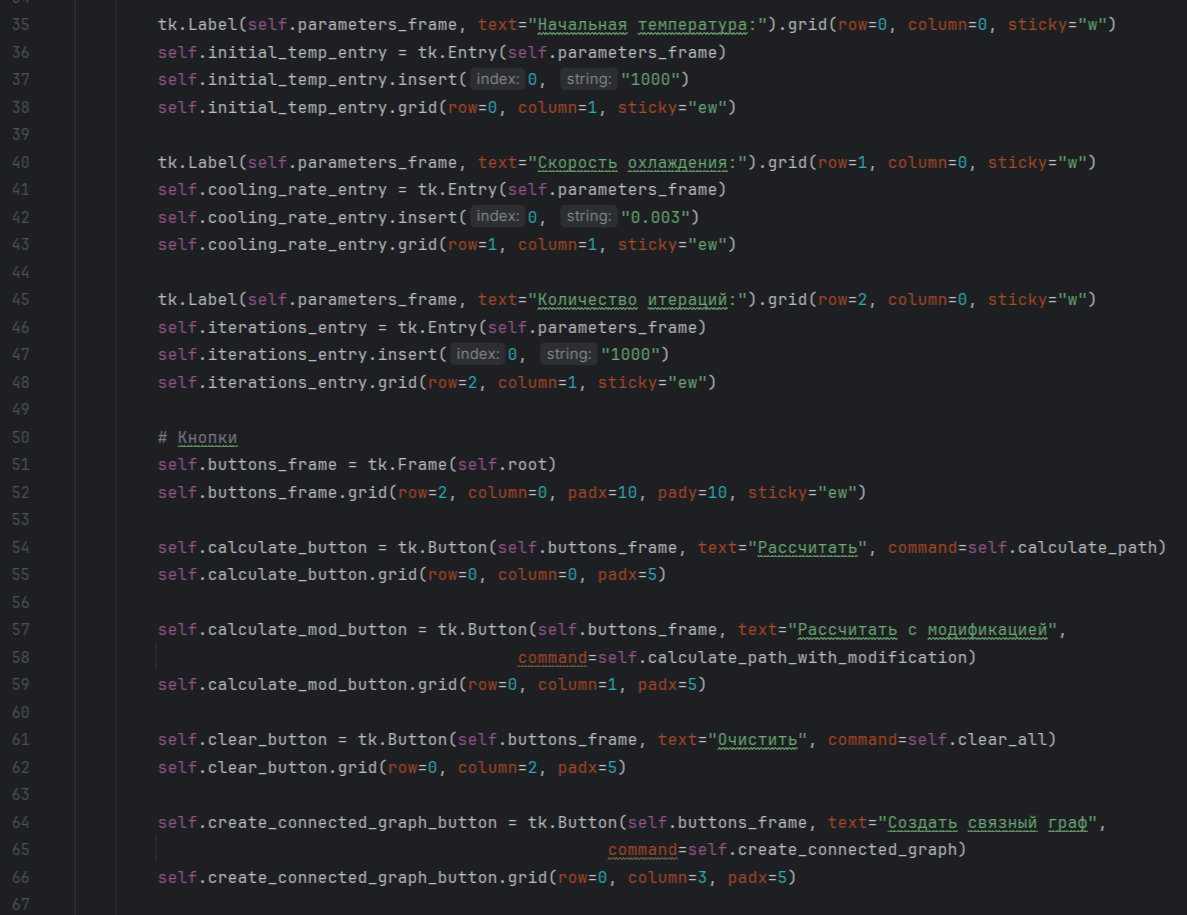
В ходе лабораторной работы был исследован алгоритм имитации отжига для решения задачи коммивояжёра. Результаты показали, что алгоритм эффективно находит приближённые решения, особенно при правильном подборе параметров (температура, скорость охлаждения). На малых графах он показывает хорошую точность, однако с ростом размера графа время работы увеличивается. Модификация алгоритма с использованием критерия Коши в некоторых случаях улучшает качество решения, но требует дополнительных вычислительных затрат.

Сравнение с методом ближайших соседей выявило: метод ближайших соседей работает быстрее, но чаще даёт менее оптимальные решения, тогда как имитация отжига обеспечивает лучшие результаты за счёт большего времени выполнения. Таким образом, выбор алгоритма зависит от конкретной задачи: для быстрых расчётов подойдёт метод ближайших соседей, а для более точных решений — имитация отжига, особенно на графах небольшого размера.

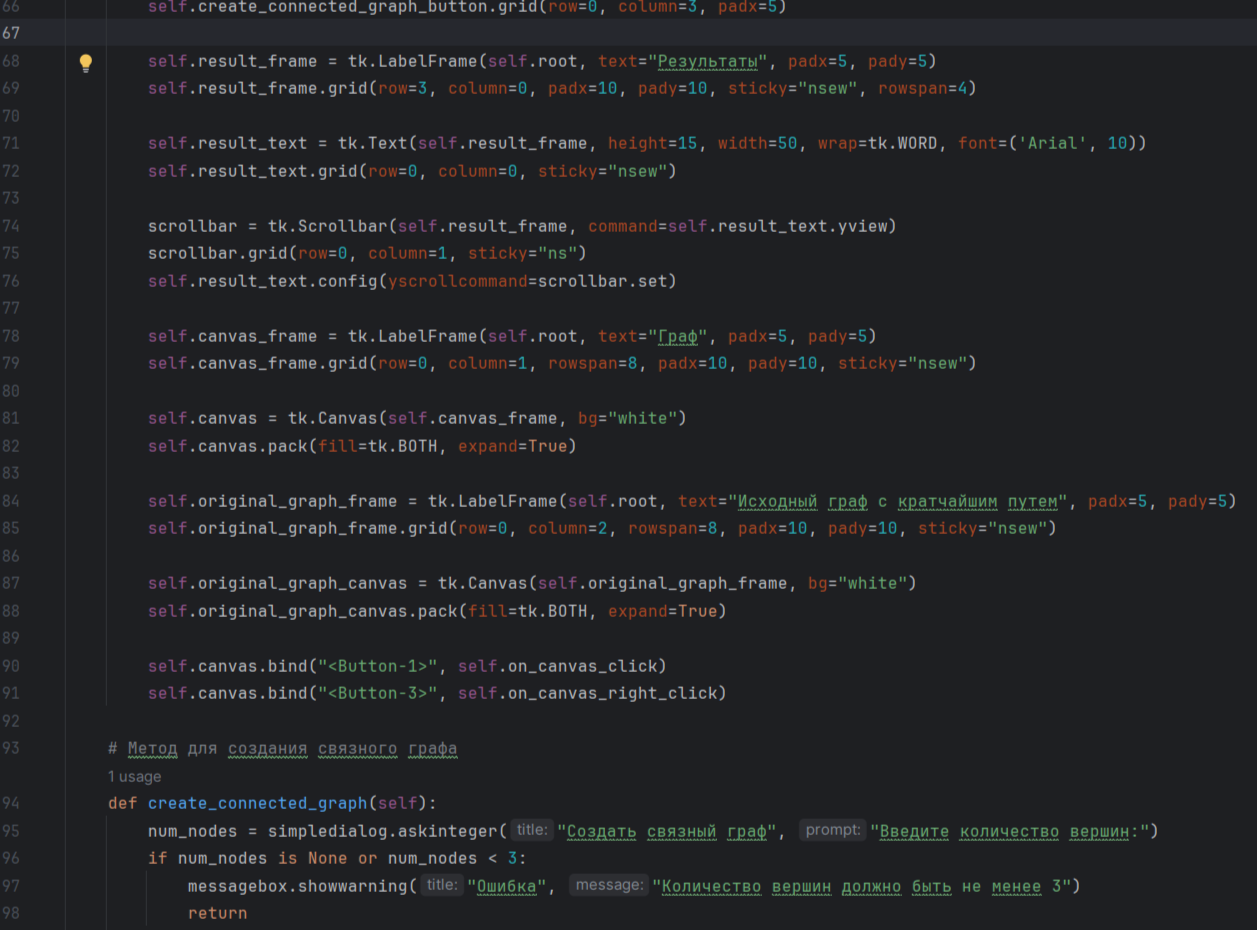
# **Листинг**



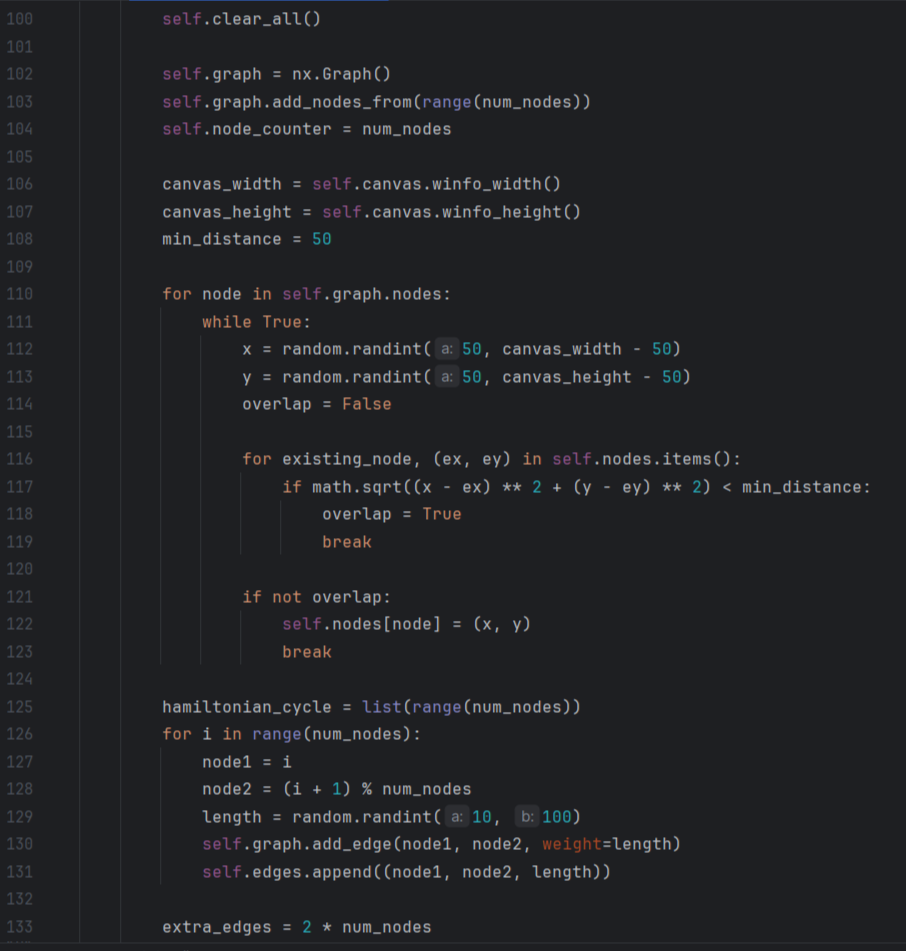
*Рис 6: Листинг*



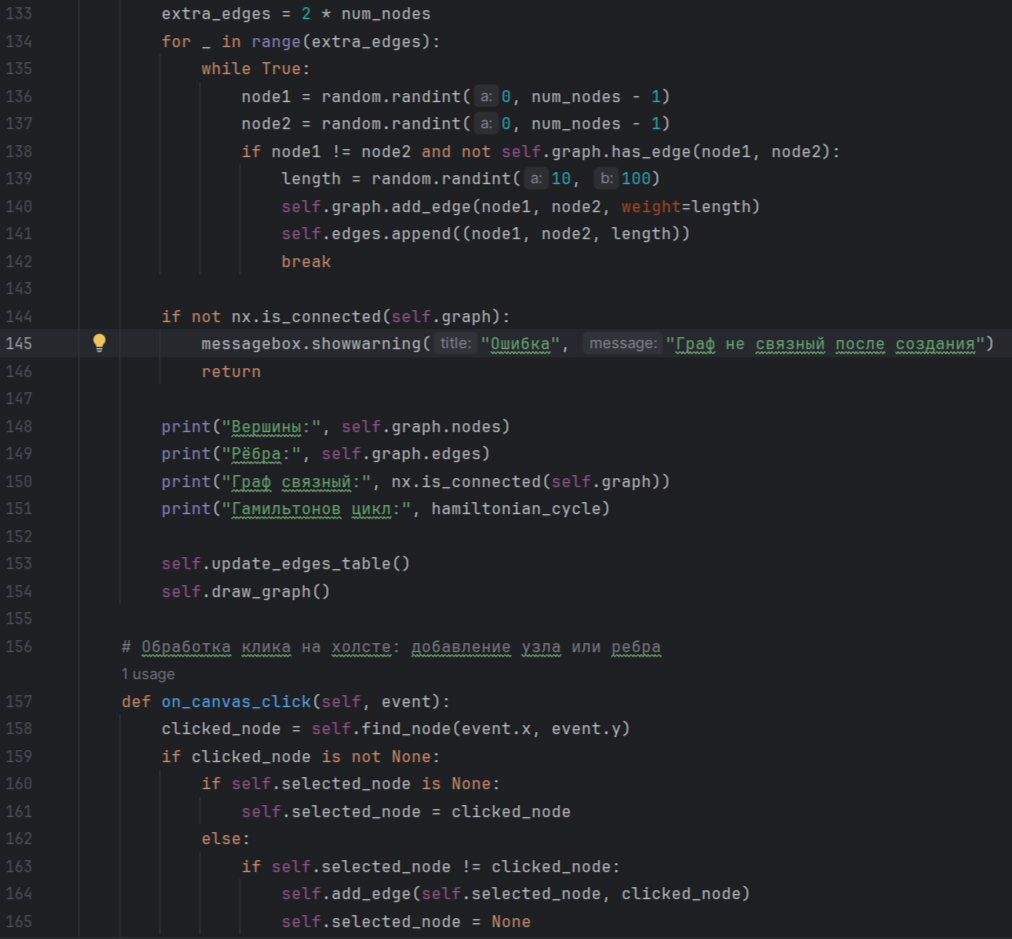
*Рис 7: Листинг*



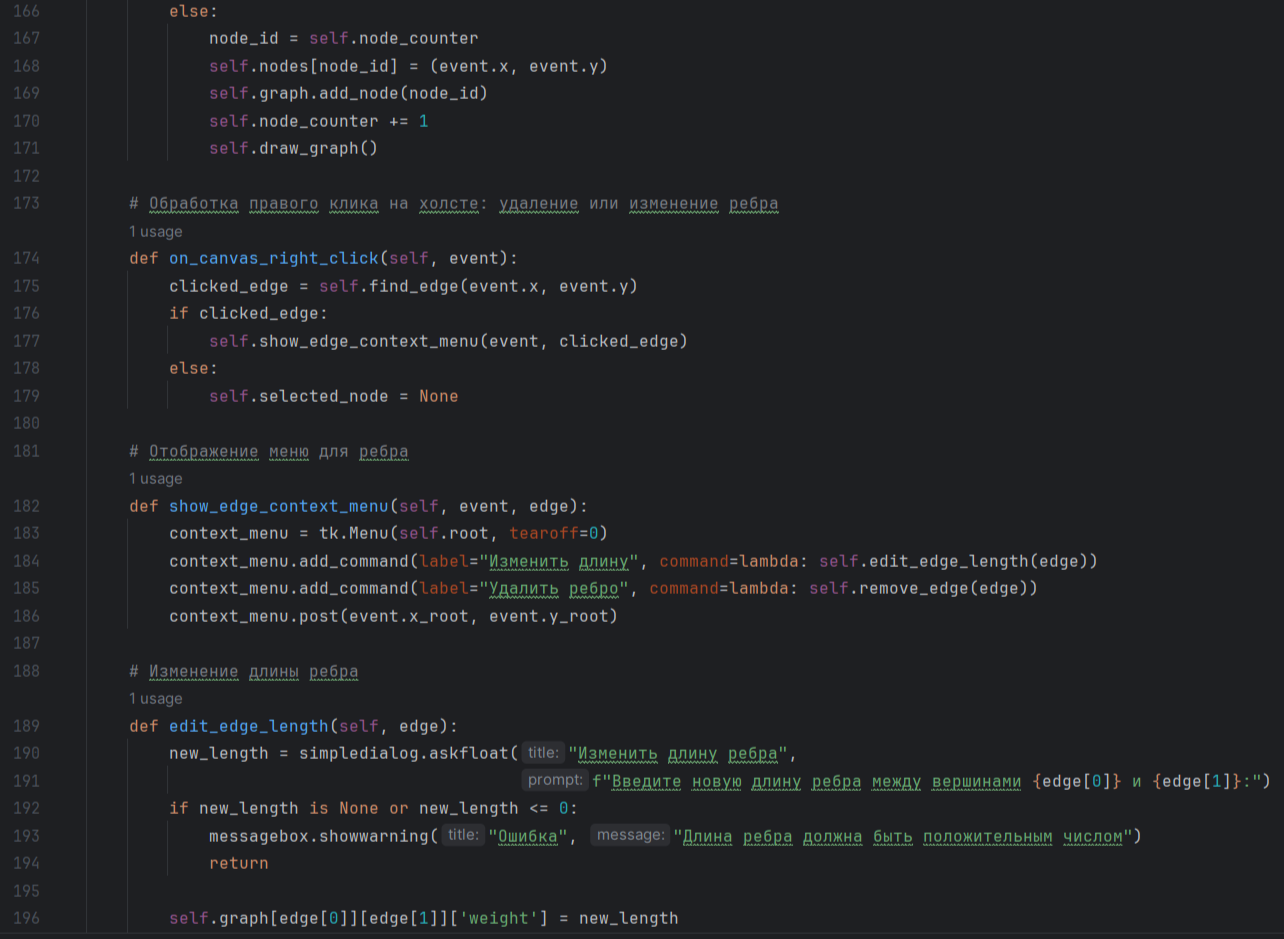
*Рис 8: Листинг*



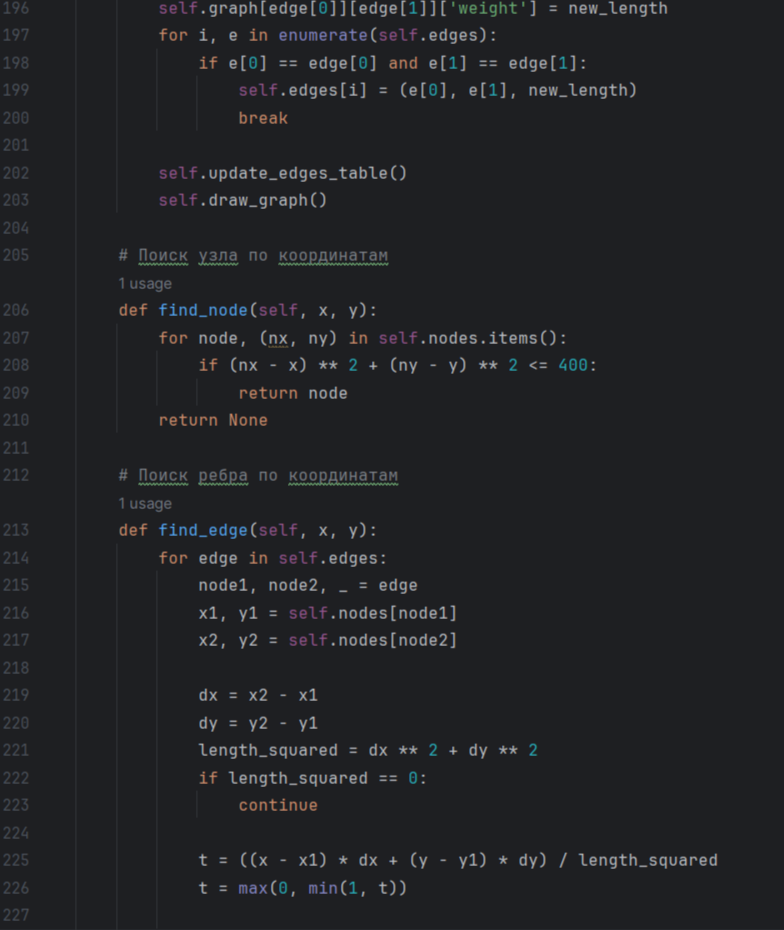
*Рис 9: Листинг*



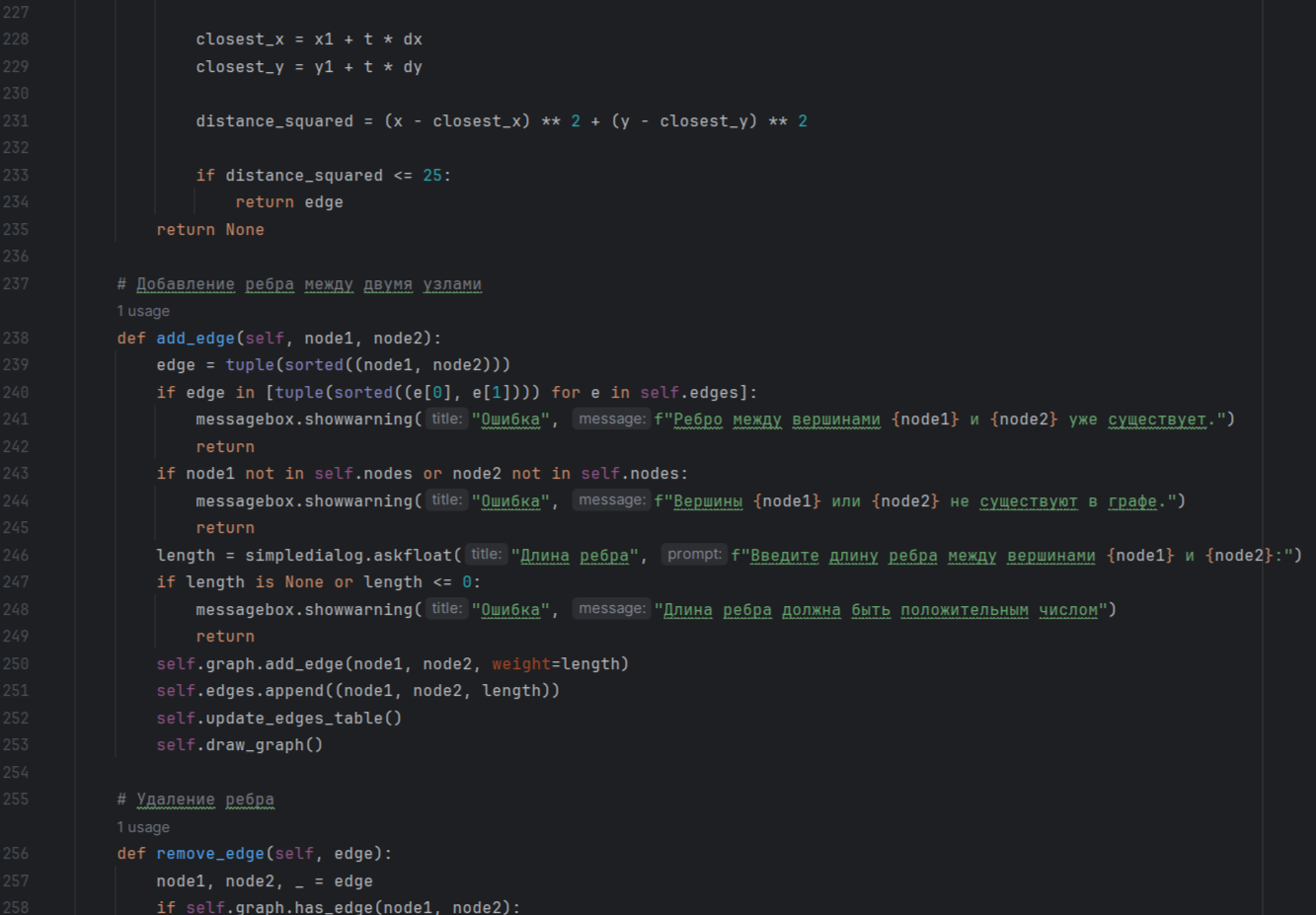
*Рис 10: Листинг*



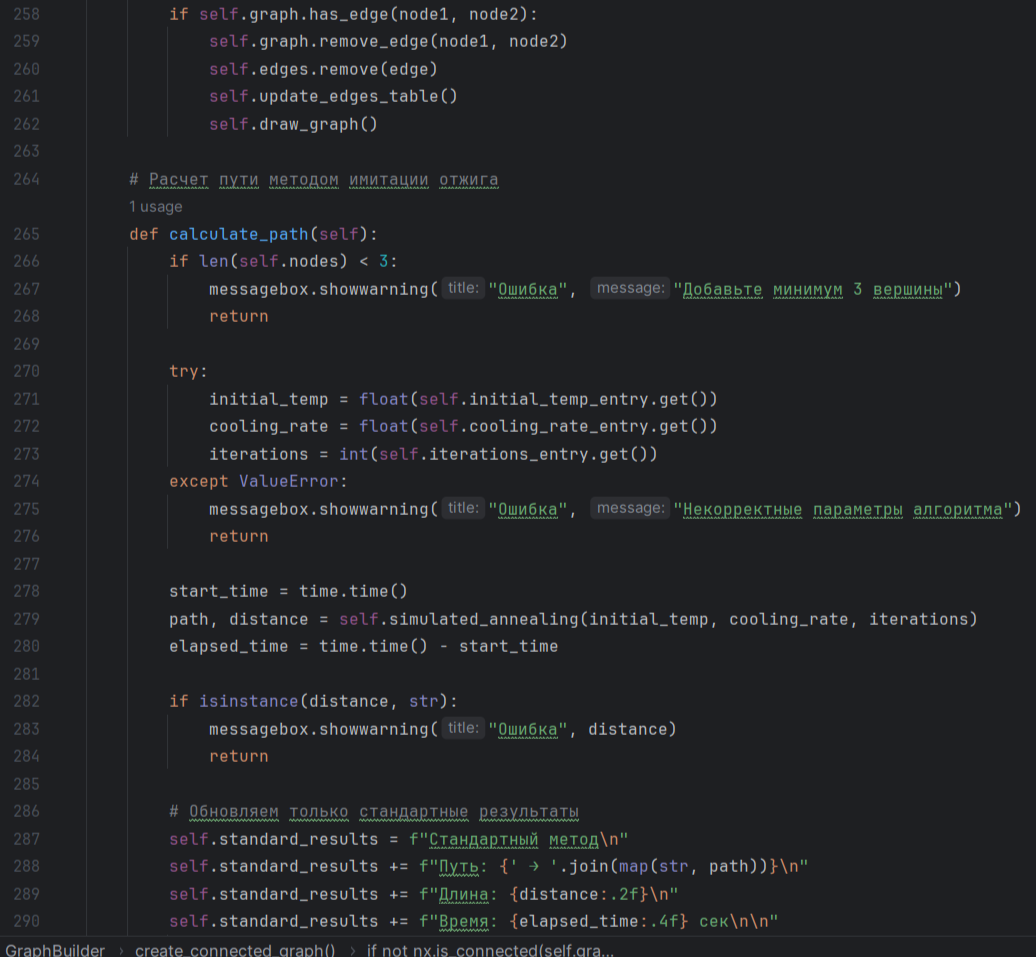
*Рис 11: Листинг*



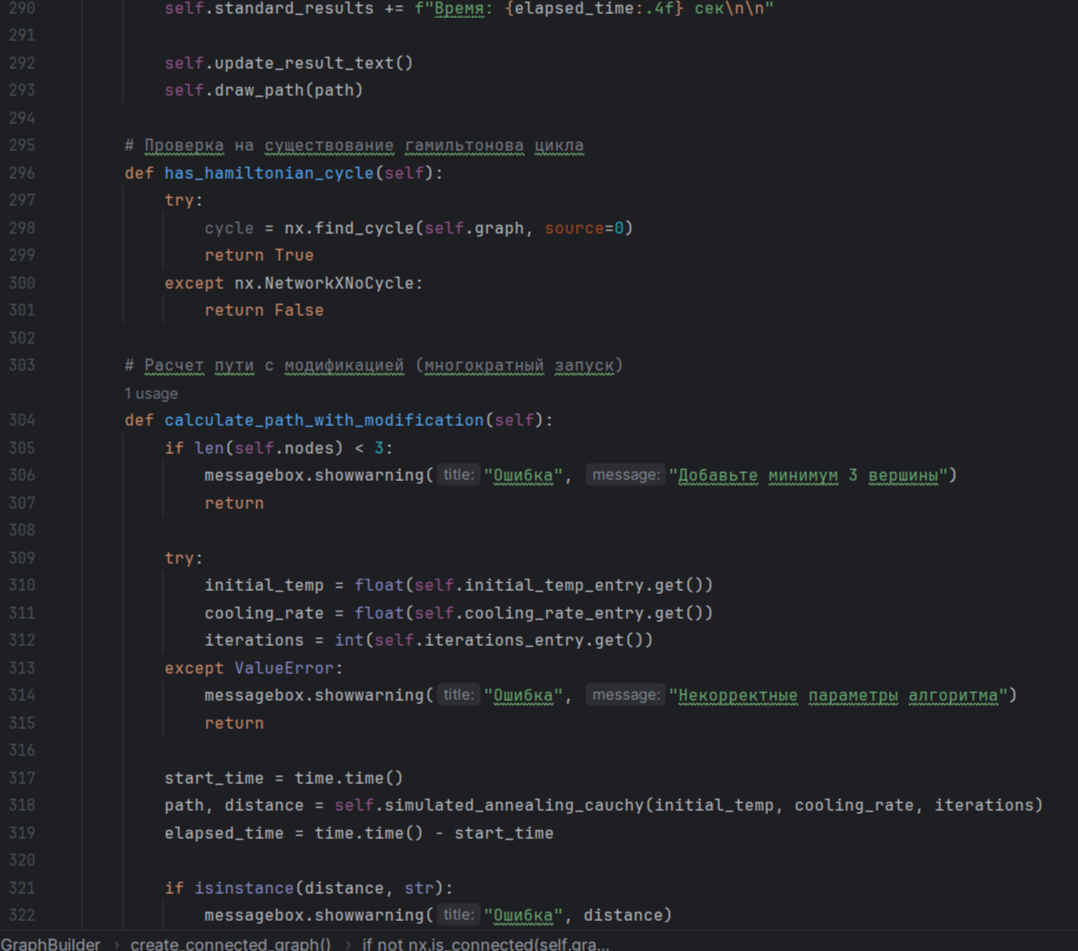
*Рис 12: Листинг*



*Рис 13: Листинг*



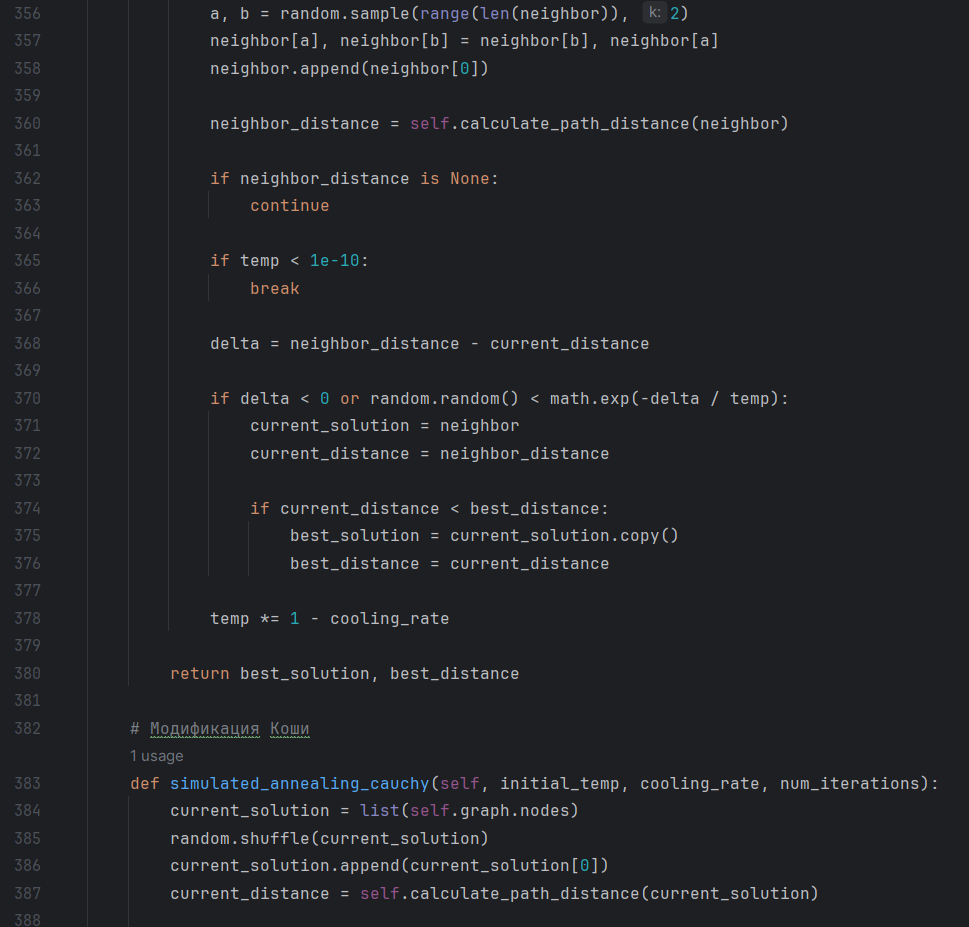
*Рис 14: Листинг*



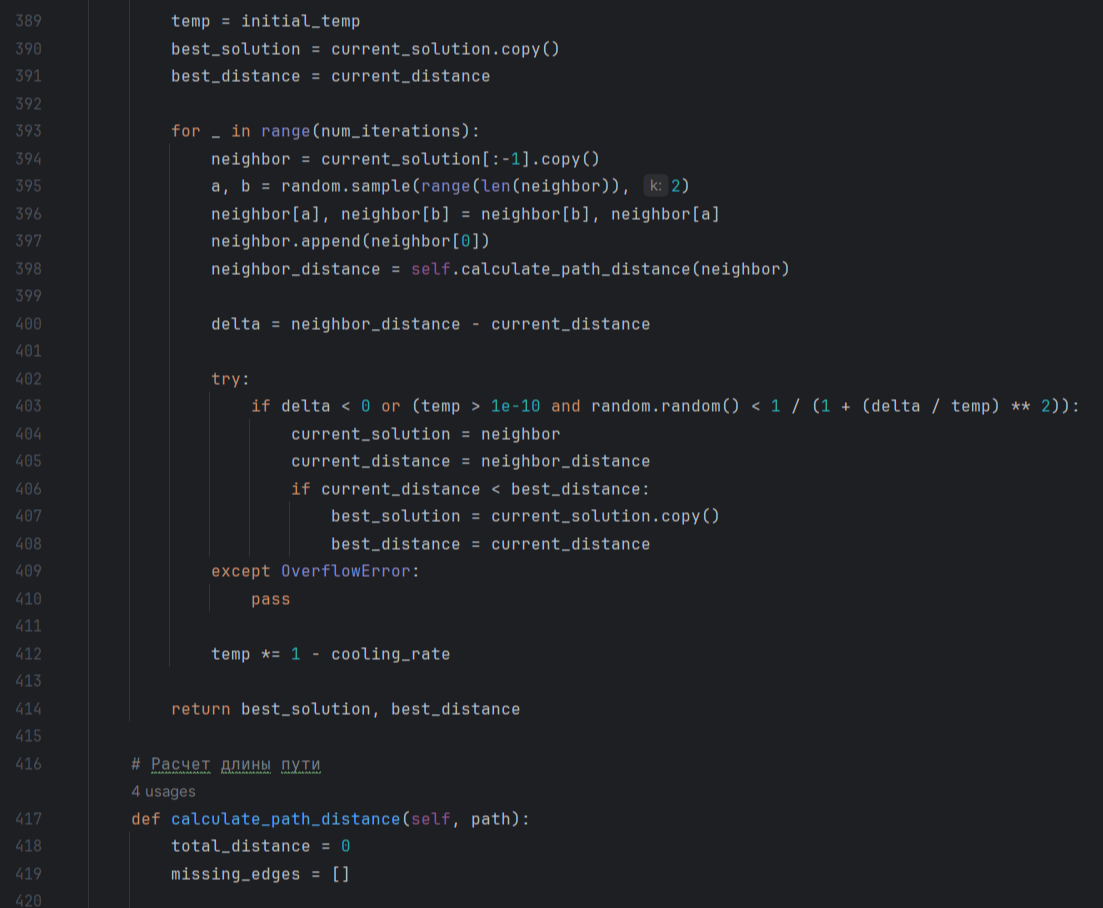
*Рис 15: Листинг*



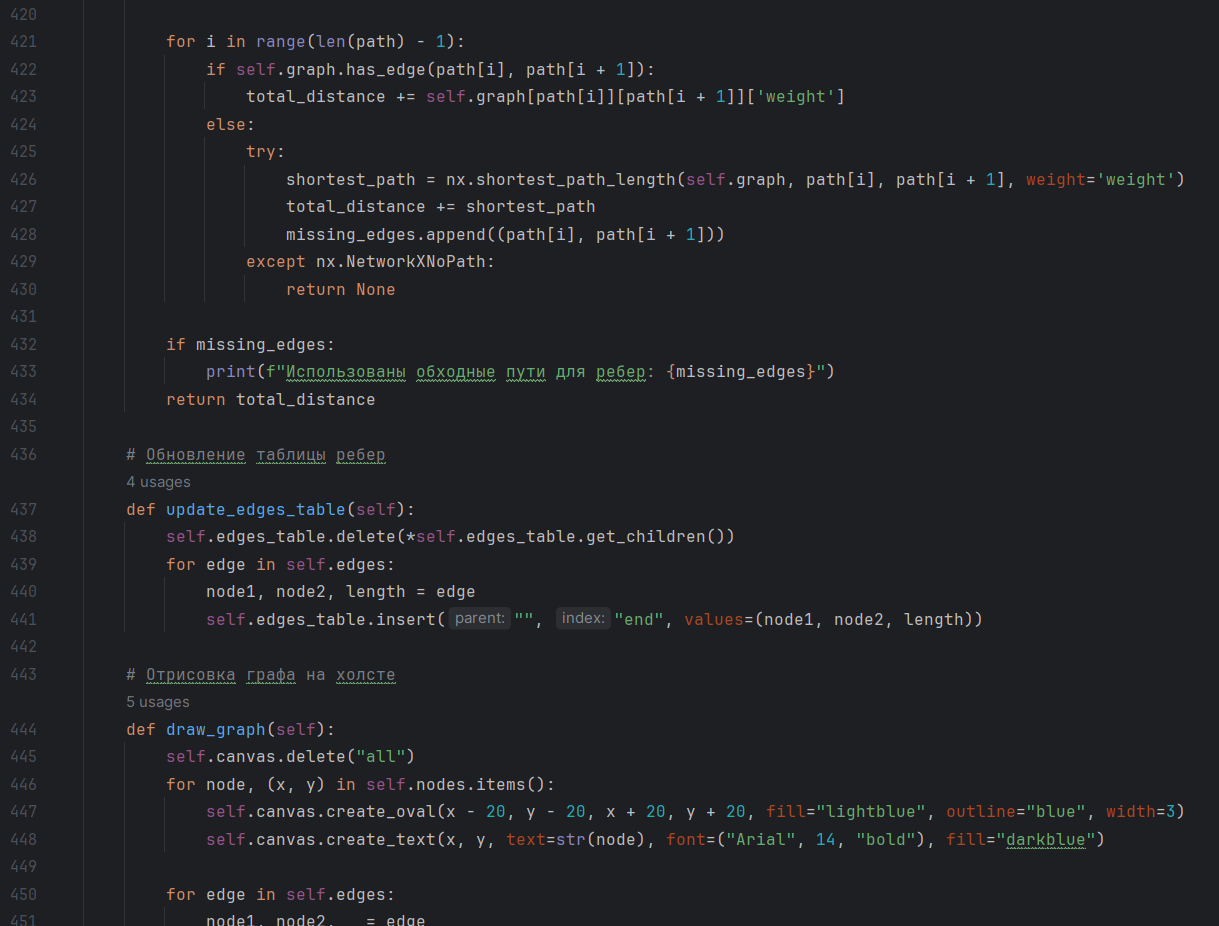
*Рис 16: Листинг*



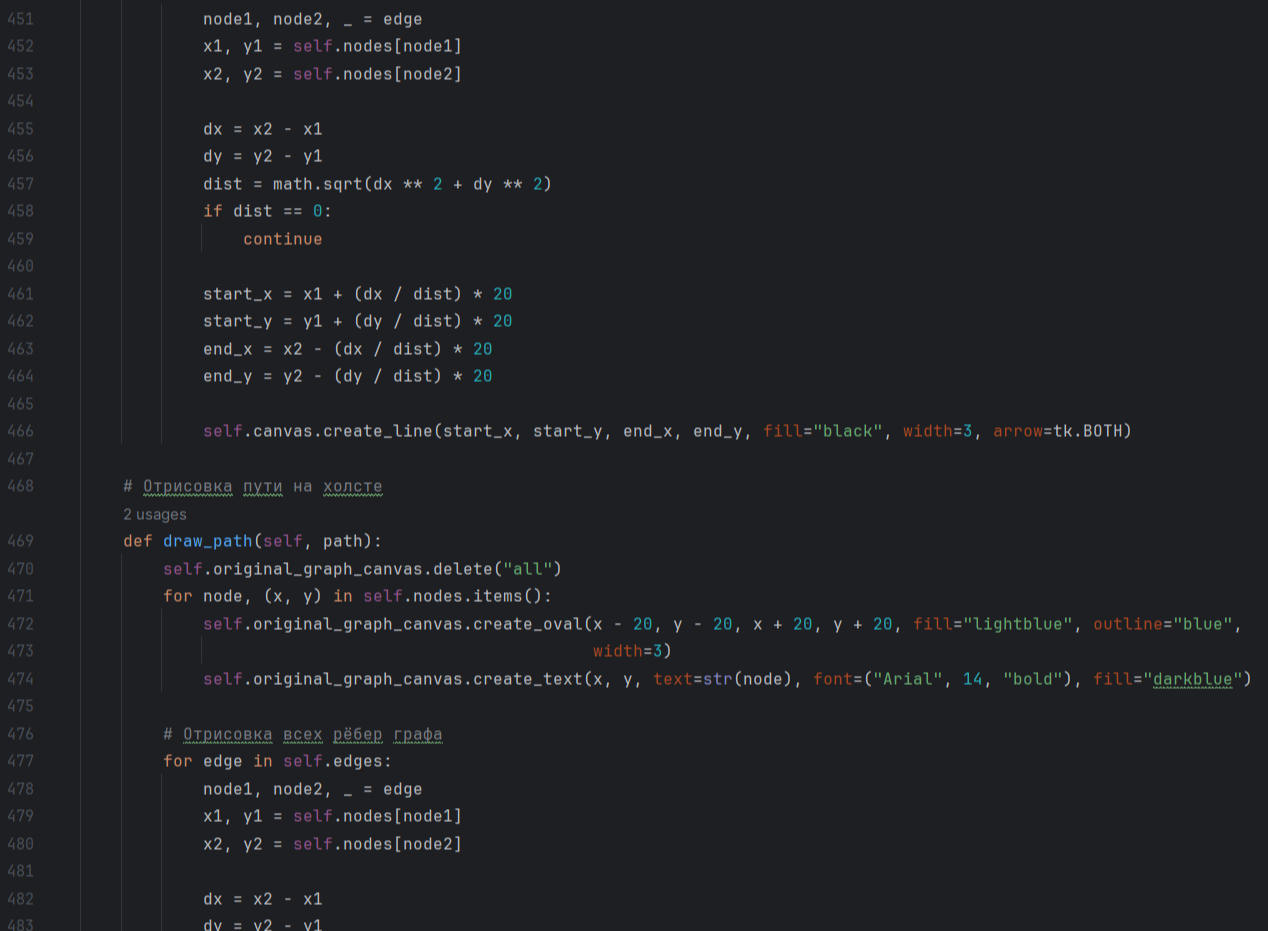
*Рис 17: Листинг*

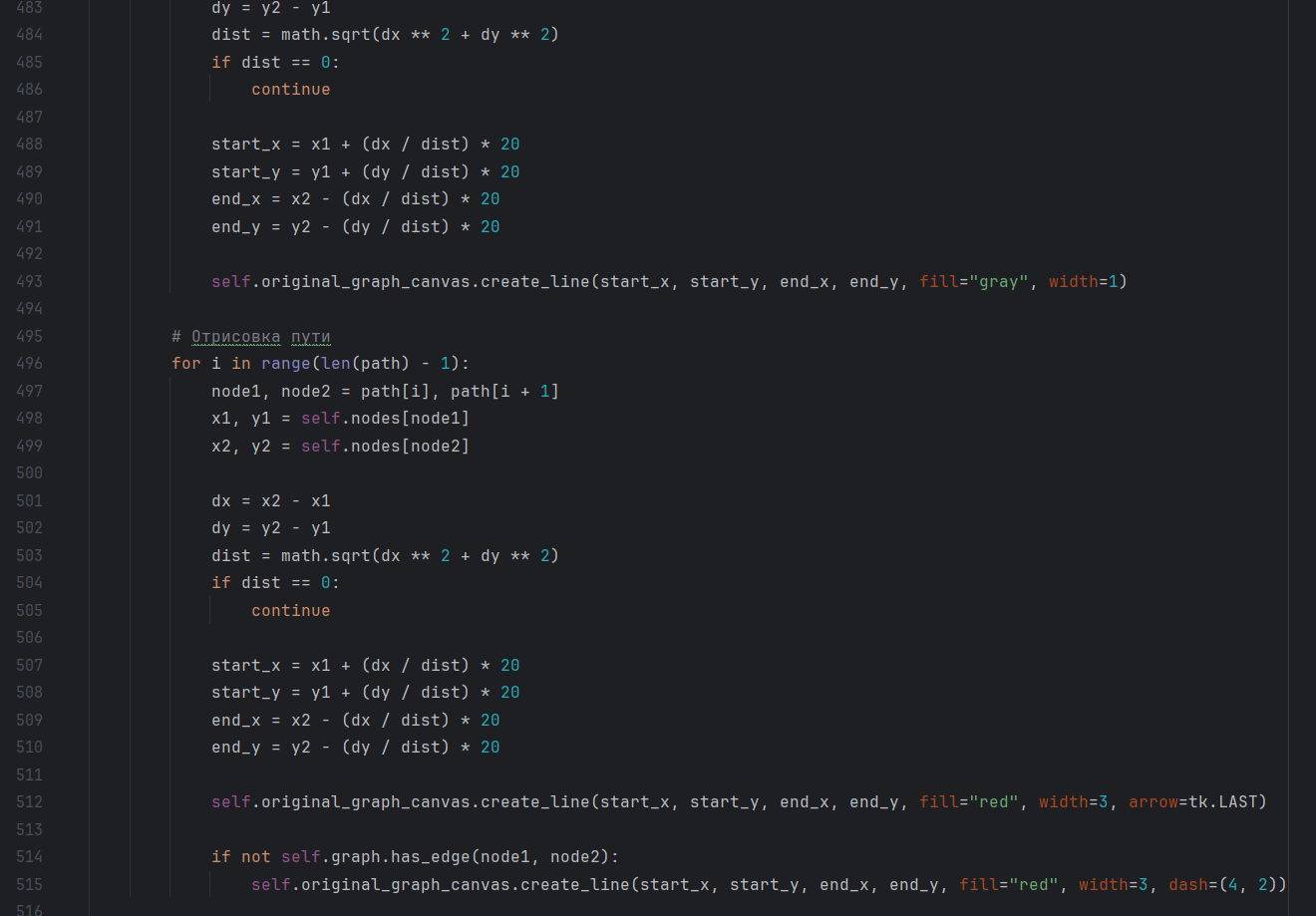


*Рис 18: Листинг*

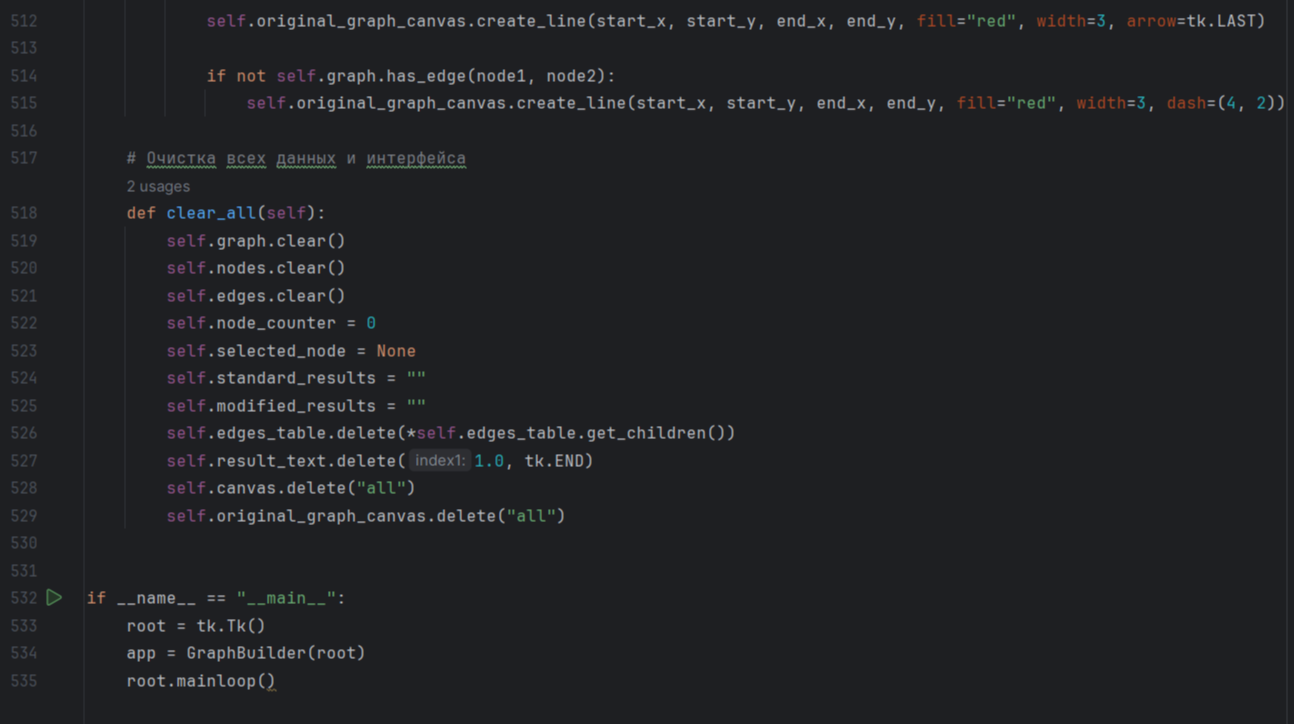


*Рис 19: Листинг*





*Рис 20: Листинг*



*Рис 21: Листинг*