**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**ОТЧЕТ**

**по лабораторной работе №3**

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

**на тему «Решение задачи о коммивояжере с помощью метода имитации муравьиной колонии»**

**Студент гр. 22Б16-пу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Олизько С.С.**

**Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дик А.Г.**

**Санкт-Петербург**

**2023 г**

Оглавление

[**Цель работы:** 3](#_Toc153215487)

[**Задача:** 3](#_Toc153215488)

[**Теоритическая часть:** 3](#_Toc153215489)

[**Алгоритм метода:** 5](#_Toc153215490)

[**Описание программы:** 6](#_Toc153215491)

[**Рекомендации для пользователя:** 6](#_Toc153215492)

[**Контрольный пример:** 8](#_Toc153215493)

[**Анализ результатов работы алгоритма и вводных условий:** 8](#_Toc153215494)

[**Вывод:** 10](#_Toc153215495)

[**Источники:** 10](#_Toc153215496)

[**Листинг программы с детальными комментариями:** 11](#_Toc153215497)

# **Цель работы:**

Исследование особенностей решения задачи о коммивояжере с использованием алгоритма имитации отжига и алгоритма ближайшего соседа, а также сравнение их результатов на контрольном примере, представленном в виде взвешенного орграфа.

# **Задачи:**

Формализовать задачу о коммивояжере с использованием алгоритма имитации отжига. Найти кратчайший гамильтонов цикл с помощью алгоритма имитации отжига для контрольного примера. Сравнить результаты решения задачи о коммивояжере с помощью алгоритма имитации отжига и алгоритма ближайшего соседа.

# **Теоретическая часть:**

Задача коммивояжера:  
  
 Задача коммивояжера (Travelling Salesman Problem, TSP) является одной из классических задач комбинаторной оптимизации, которая заключается в нахождении самого короткого маршрута, проходящего через все города из некоторого заданного списка, посещая каждый город ровно один раз, и возвращаясь в исходный город. В контексте задачи, каждый город рассматривается как вершина графа, а расстояние между городами - весом ребра, соединяющего их.  
  
 Гамильтонов цикл:  
  
 Гамильтонов цикл в графе - это такой цикл, который проходит через каждую вершину графа ровно один раз и возвращается в начальную вершину. В задаче коммивояжера идеальное решение представляет собой гамильтонов цикл минимальной длины.  
  
 Метод имитации муравьиной колонии:  
  
 Метод имитации муравьиной колонии (Ant Colony Optimization, ACO) - это метаэвристический алгоритм, вдохновленный поведением муравьев в природе. Муравьи обладают способностью находить кратчайший путь к источнику пищи, оставляя за собой феромонные следы. В алгоритме ACO решение ищется путем моделирования этого поведения муравьев.  
  
 Основные шаги алгоритма ACO:  
 1. Инициализация: Распределение феромона на ребрах графа.  
 2. Построение решения: Муравьи выбирают путь по вершинам графа, учитывая как расстояние между вершинами, так и количество феромона на ребрах.  
 3. Обновление феромона: Феромон на ребрах обновляется в зависимости от эффективности пройденных маршрутов.  
 4. Повторение: Процесс построения решений и обновления феромона повторяется до достижения критерия останова.  
  
 Недостатки метода имитации муравьиной колонии:  
  
 1. Склонность к застреванию в локальных оптимумах:При недостаточной разнообразности исследуемых путей муравьи могут застрять в локальных оптимумах, не достигнув глобального оптимального решения.  
 2. Чувствительность к параметрам: Эффективность алгоритма сильно зависит от настройки его параметров, таких как коэффициент испарения феромона и коэффициенты влияния расстояния и феромона на выбор пути.   
 3. Вычислительная сложность: Для достижения хорошего качества решения может потребоваться большое количество итераций, что приводит к высокой вычислительной сложности алгоритма.

# **Алгоритм метода:**

# **Инициализация:**

* + Создание колонии муравьев, каждый из которых начинает свой путь из случайной вершины графа.
  + Задание начальных значений феромонов на ребрах графа.

1. **Обновление феромонов:**
   * Для каждого ребра в графе уменьшается количество феромона на нем в соответствии с коэффициентом испарения.
   * Для каждого муравья, завершившего свой путь, увеличивается количество феромона на пройденных им ребрах в зависимости от эффективности его маршрута.
2. **Выбор следующей вершины:**
   * Для каждой вершины, доступной муравью для посещения, вычисляется вероятность выбора этой вершины с учетом количества феромона на ребре и длины ребра.
   * Выбор следующей вершины осуществляется случайным образом с учетом вероятностей.
3. **Поиск решения:**
   * Муравьи последовательно выбирают следующие вершины до тех пор, пока не посетят все вершины графа.
   * Если муравей обнаруживает, что больше нет доступных вершин для посещения и он не обошел все вершины, он завершает свой путь, считая текущий маршрут бесконечно длинным.
4. **Итерации:**
   * Алгоритм повторяется заданное количество итераций.
   * На каждой итерации феромоны обновляются, а лучший найденный муравьем маршрут сохраняется.
5. **Выбор лучшего результата:**
   * По завершении всех итераций выбирается лучший муравьин маршрут, имеющий наименьшую длину.
6. **Визуализация:**
   * Если гамильтонов цикл найден, он отображается на графическом холсте.
   * В противном случае выводится сообщение о его отсутствии.

# **Описание программы:**

Программа реализована на языке Python 3.12 с использованием следующих пакетов: tkinter, networkx и matplotlib.pyplot, numpy, random, math, threading. В программе используются функций. В таблицах описание функций программы.

*Таблица 1: функции*

| Имя функции | Тип возвращаемого значения | Описание функции |
| --- | --- | --- |
| Ant | None | Конструктор класса Ant. Инициализирует муравья, выбирая случайную начальную вершину и создавая путь с этой вершиной. |
| can\_visit | bool | Проверяет, можно ли посетить указанную вершину, учитывая текущий путь муравья и наличие ребра в графе между текущей вершиной и указанной вершиной. |
| visit | None | Добавляет указанную вершину в путь муравья и увеличивает длину пути на вес ребра между текущей вершиной и указанной вершиной. |
| get\_cycle\_length | float | Возвращает длину цикла муравья. Если муравей посетил все вершины и вернулся в начальную вершину, возвращает длину пути, иначе возвращает бесконечность. |
| **str** | str | Возвращает строковое представление муравья, содержащее информацию о его пути, длине пути и длине цикла. |
| AntColony | None | Конструктор класса AntColony. Инициализирует муравьиную колонию, создавая указанное количество муравьев и задавая начальные значения феромонов на ребрах графа. |
| update\_pheromones | None | Обновляет значения феромонов на ребрах графа в соответствии с формулой обновления феромонов для каждого прошедшего муравья. |
| choose\_next\_node | int or None | Выбирает следующую вершину для посещения муравьем на основе вероятности, зависящей от количества феромонов на ребре и веса ребра. Если все вершины уже посещены, возвращает None. |
| search\_serial | Ant | Выполняет последовательный поиск оптимального маршрута для муравьиной колонии. Возвращает лучшего муравья с наименьшей длиной цикла. |
| search\_parallel | Ant | Выполняет параллельный поиск оптимального маршрута для муравьиной колонии. Возвращает лучшего муравья с наименьшей длиной цикла. |
| search | Ant | Выполняет поиск оптимального маршрута для муравьиной колонии в зависимости от выбранного режима (последовательный или параллельный). |
| ant\_hamiltonian\_cycle | nx.DiGraph or None | Выполняет поиск гамильтонова цикла с использованием муравьиной колонии и визуализирует результат на графическом холсте. Возвращает граф гамильтонова цикла или None, если не найден. |

# **Рекомендации для пользователя:**

Запуск программы:

1. Убедитесь, что на вашем компьютере установлен Python.
2. Установите необходимые библиотеки: networkx, tkinter и threading. Выполните команду pip install networkx tkinter threading.
3. Запустите программу для активации графического интерфейса.

Использование программы:

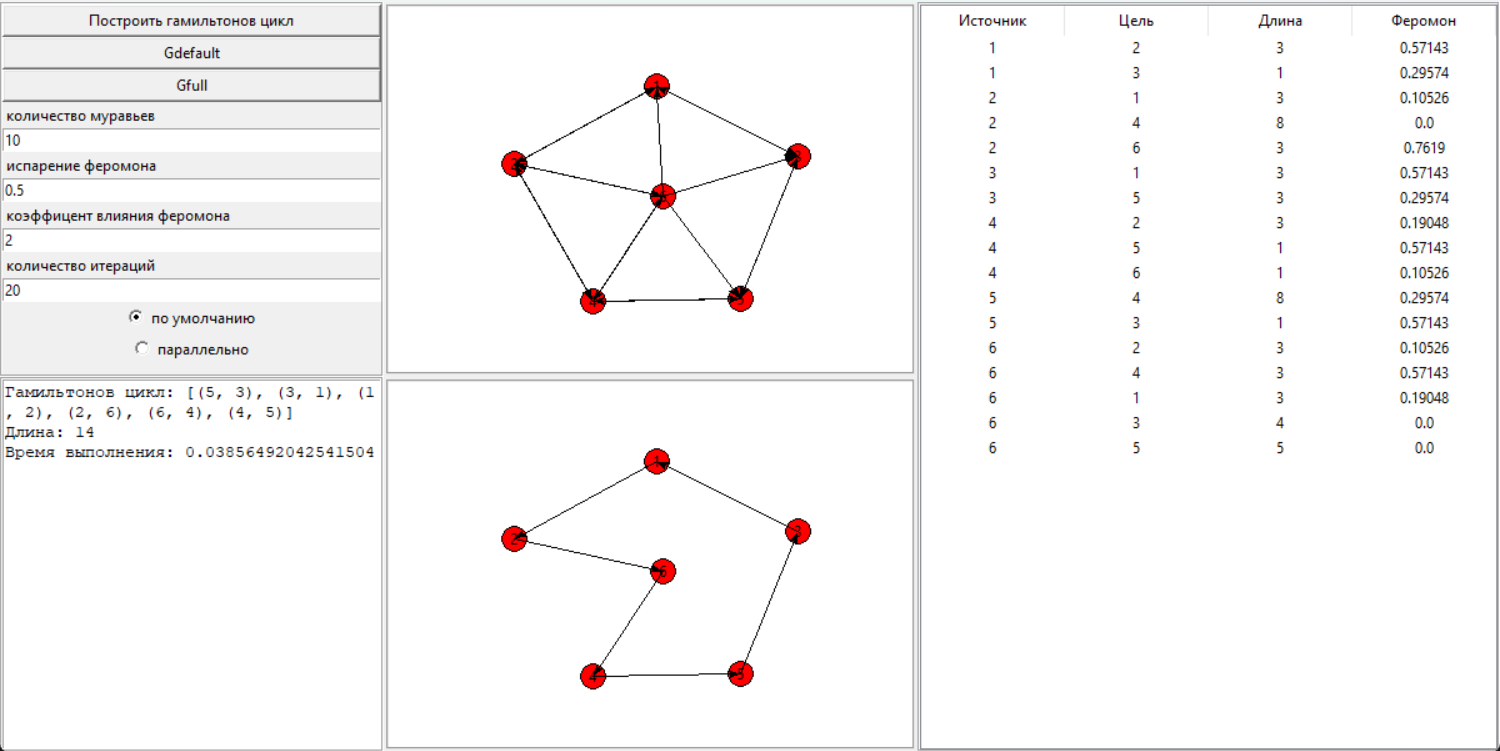
1. Используйте холст для создания вершин графа левой кнопкой мыши и добавления ребер правой кнопкой мыши.
2. Введите значения параметров алгоритма (число муравьев, коэффициент испарения феромона, влияние феромона, количество итераций) в соответствующие поля ввода.
3. Выберите режим работы муравьиной колонии: последовательный или параллельный.
4. Нажмите кнопку "Построить гамильтонов цикл", чтобы запустить поиск гамильтонова цикла.
5. Результаты будут отображены в текстовом поле и на холсте.

Рекомендуется убедиться в правильности ввода значений параметров перед запуском алгоритма поиска гамильтонова цикла.

**Исходный код программы доступен по ссылке ниже:**

<https://github.com/StephanOlizko?tab=repositories>

# **Контрольный пример:**



*Рис 1: пример окна программы*

# **Анализ результатов работы алгоритма и вводных условий:**

В обычном режиме работы алгоритм всегда сходится, и показывает хорошее среднее время работы.

Модификация алгоритма заключается в параллельном просчете путей муравьев на каждой итерации, так как они формально не зависят друг от друга. Данный подход позволяет добиться значительного улучшения времени работы алгоритма, вплоть до n раз, где n – число процессоров на устройстве.

В сравнении с алгоритмом ближайшего соседа и алгоритмом отжига, муравьиный алгори тм в стандартной реализации показывает лучший процент сходимости, однако показывает небольшое отставание по времени работы.

При тестировании использовались полный граф размера 50 и разреженный граф размера 25.

*Таблица 2: анализ*

| Алгоритм | Тип | Полносвязный, сходимость | Слабосвязный, сходимость | Полносвязный, cредняя длина и время | Слабосвязный, cредняя длина и время |
| --- | --- | --- | --- | --- | --- |
| Алгоритм муравьиной колонии | Стандартный | 1.0 | 1.0 | 65.78, 0.3667 | 73.0, 0.0745 |
| Алгоритм муравьиной колонии | Оптимизированный | 1.0 | 1.0 | 63.88, 0.5857 | 67.38, 0.0867 |
| Алгоритм отжига | Стандартный | 1.0 | 0.1 | 73.79, 0.0011 | 67.6, 0.00015 |
| Алгоритм отжига | Оптимизированный | 1.0 | 0.15 | 73.27, 0.0011 | 62.93, 0.00077 |
| Алгоритм ближайшего соседа | Стандартный | 1.0 | 0.86 | 61.97, 0.0431 | 60.34, 0.0041 |

# **Вывод:**

В ходе выполнения лабораторной работы была рассмотрена задача о коммивояжере и реализовано её решение с использованием алгоритма имитации муравьиной колонии. Этот эвристический алгоритм позволяет найти приближенное решение задачи, находя кратчайший гамильтонов цикл в графе.

Также была представлена модификация алгоритма имитации муравьиной колонии которая реализует параллельное выполнение. Это позволяет значительно улучшить время работы алгоритма.

Было проведено сравнение алгоритма ближайшего соседа, алгоритма имитации отжига и муравьиного алгоритма, а также их модификаций.

Таким образом, выполнение лабораторной работы позволило лучше понять принципы работы алгоритма имитации муравьиной колонии и его модификации, а также изучить их эффективность в решении задачи о коммивояжере.

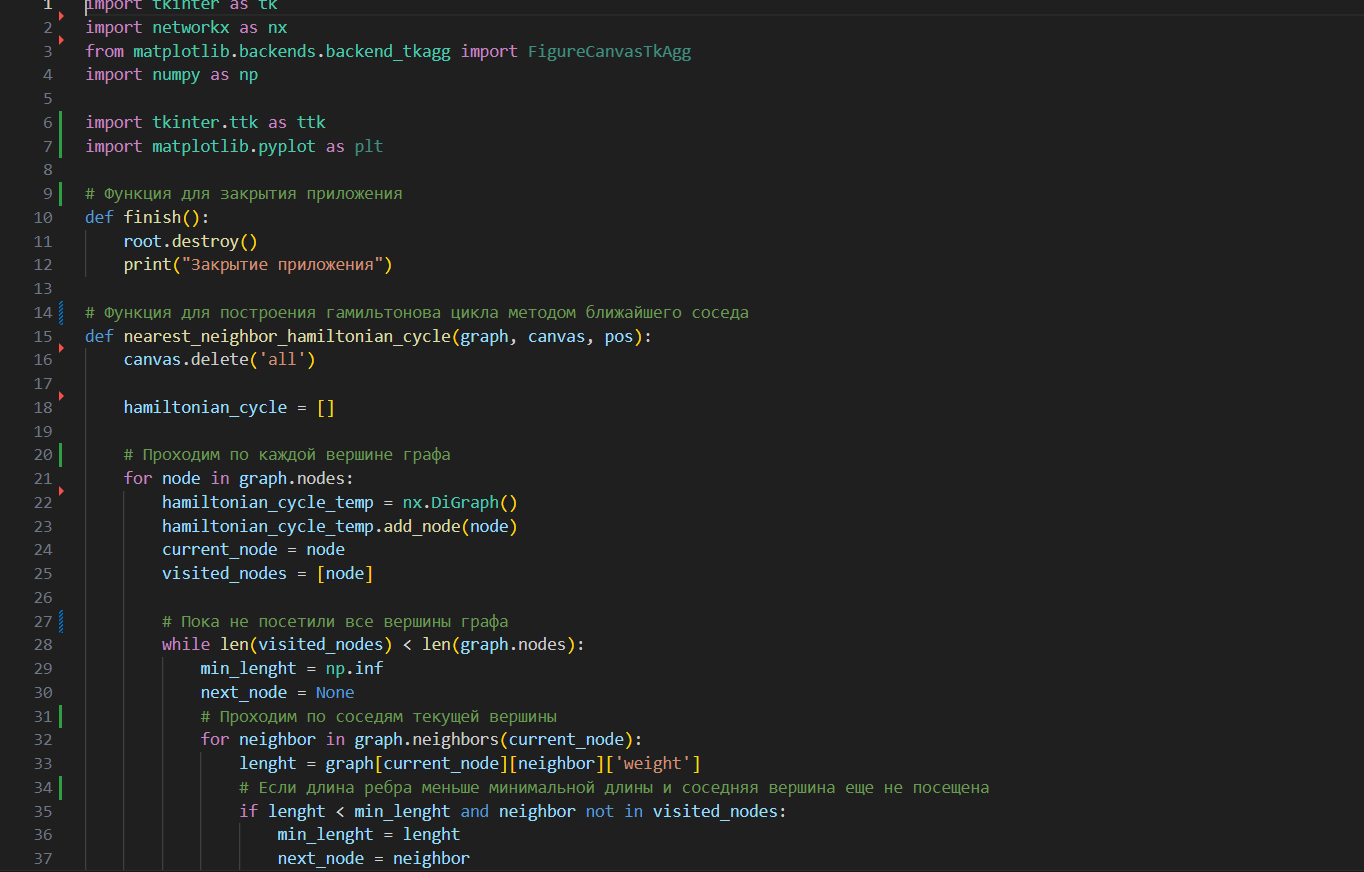
# **Источники:**

https://docs.python.org/3/library/tkinter.html

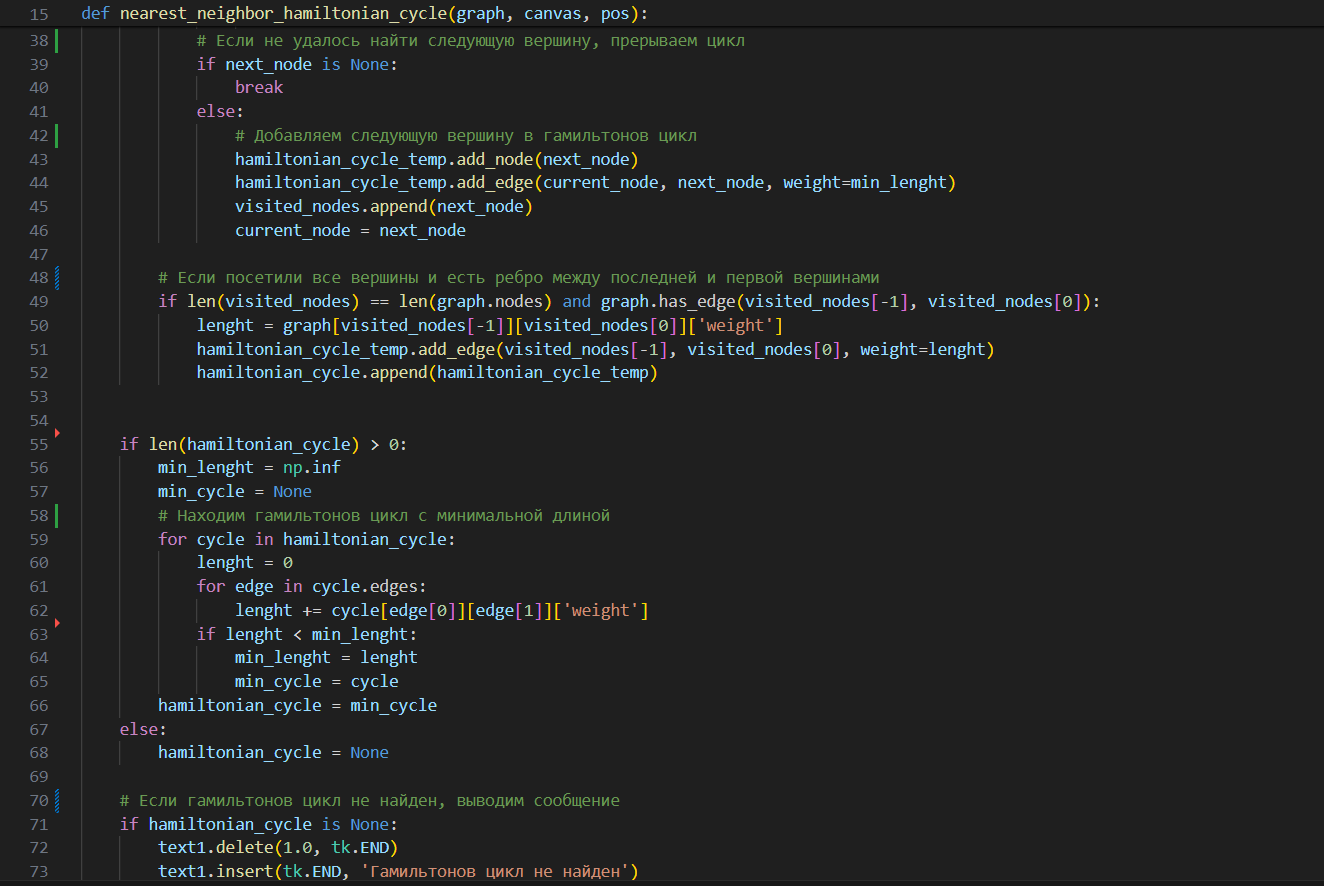
https://matplotlib.org/stable/index.html

https://numpy.org/doc/

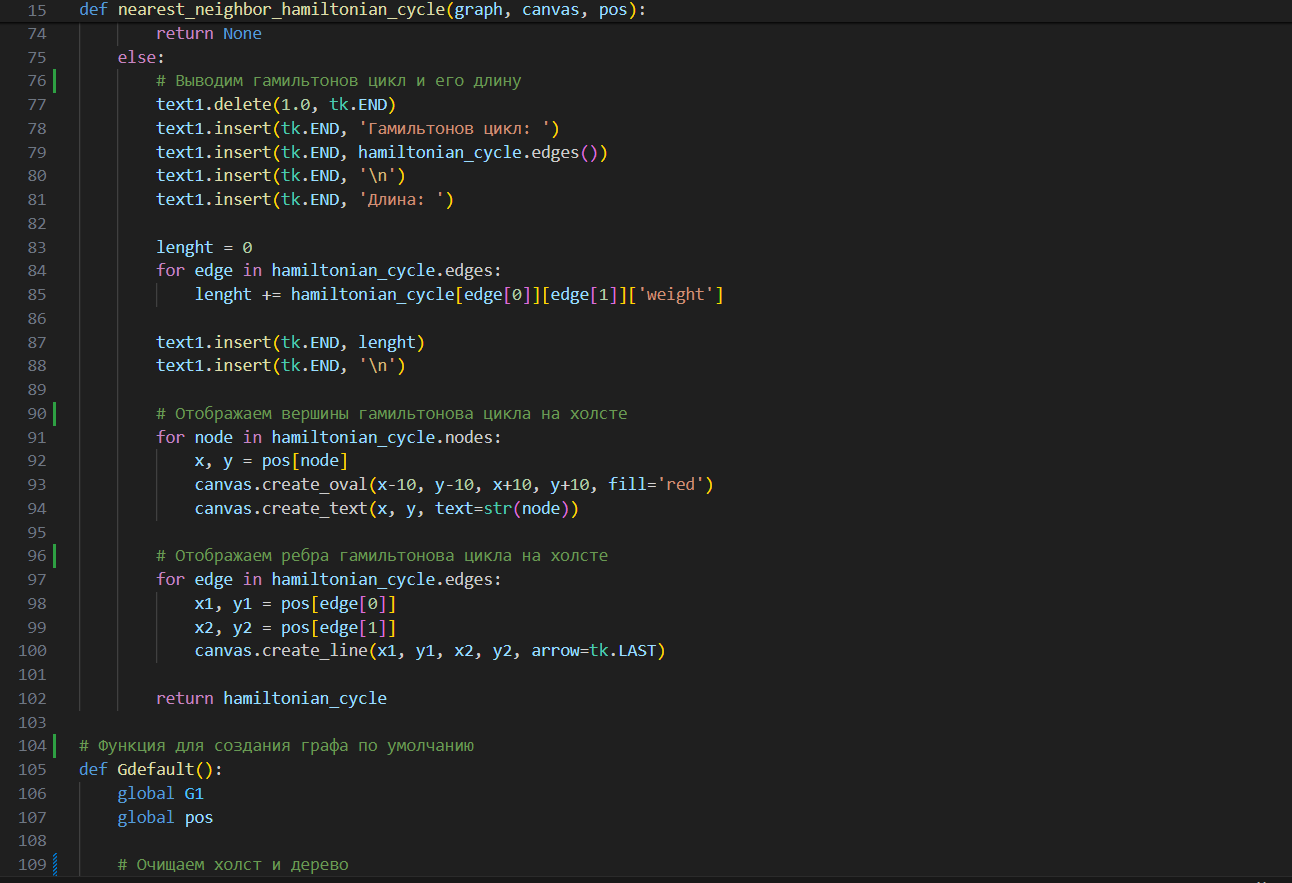
# **Листинг программы с детальными комментариями:**



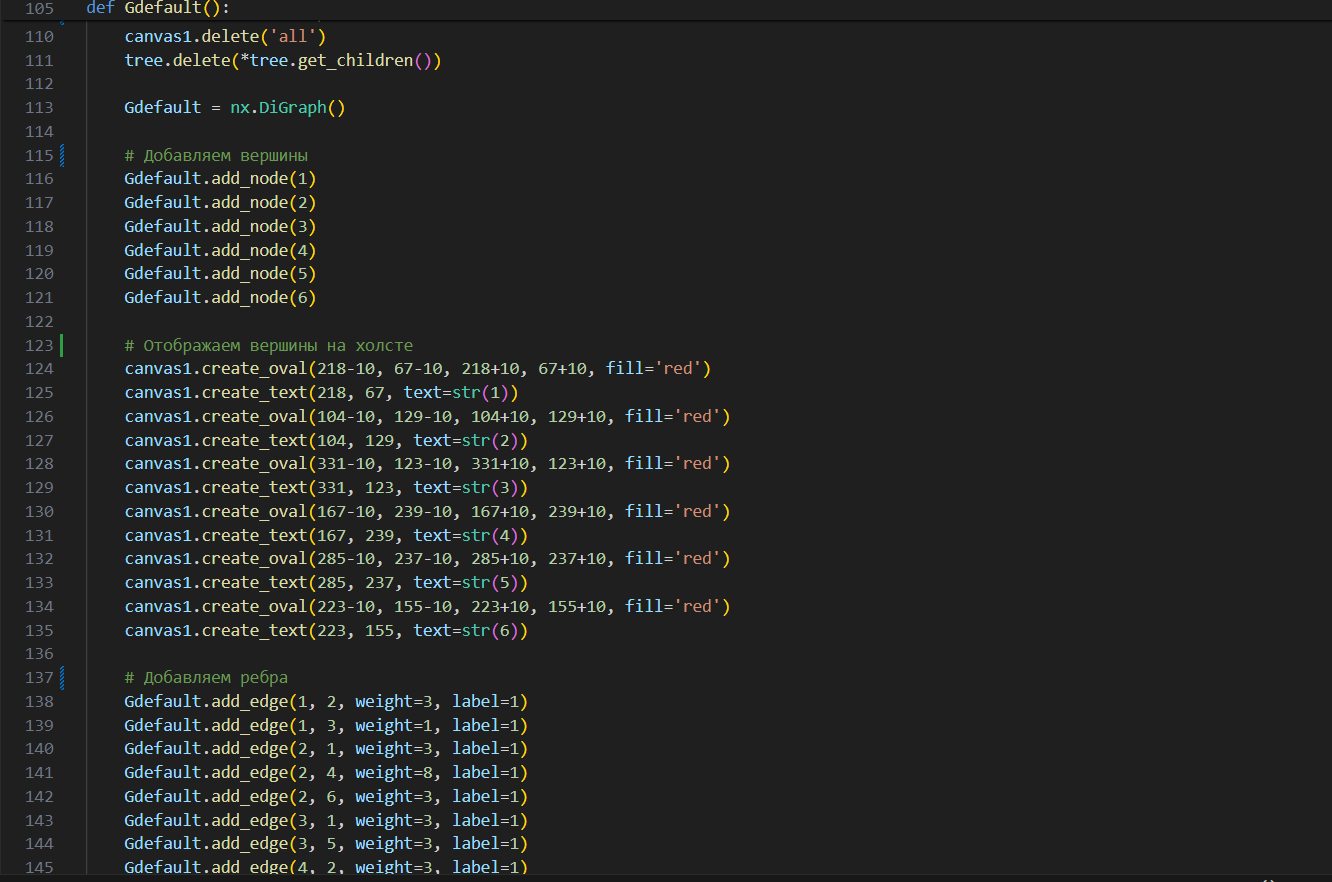
*Рис 2: листинг*



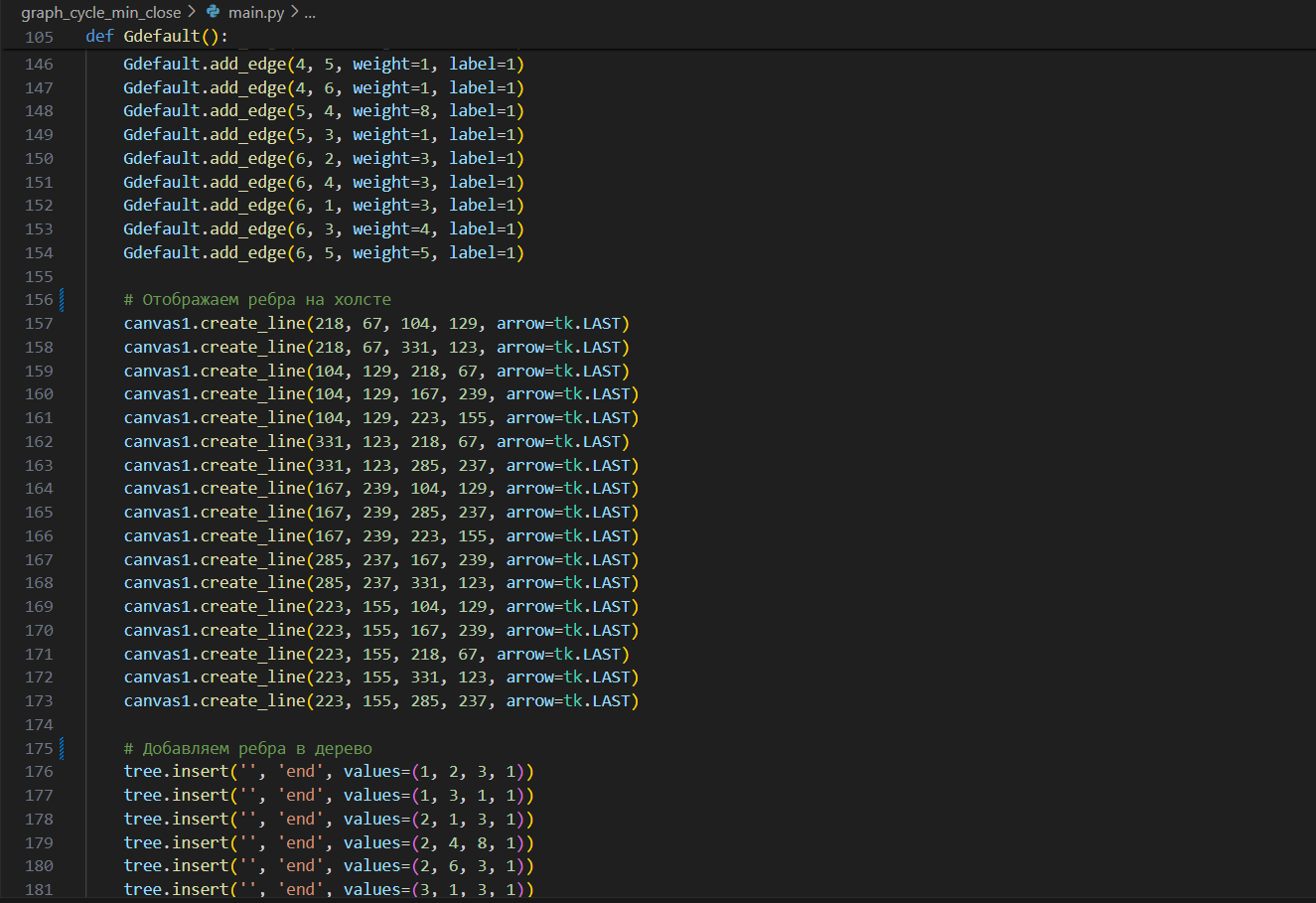
*Рис 3: листинг*



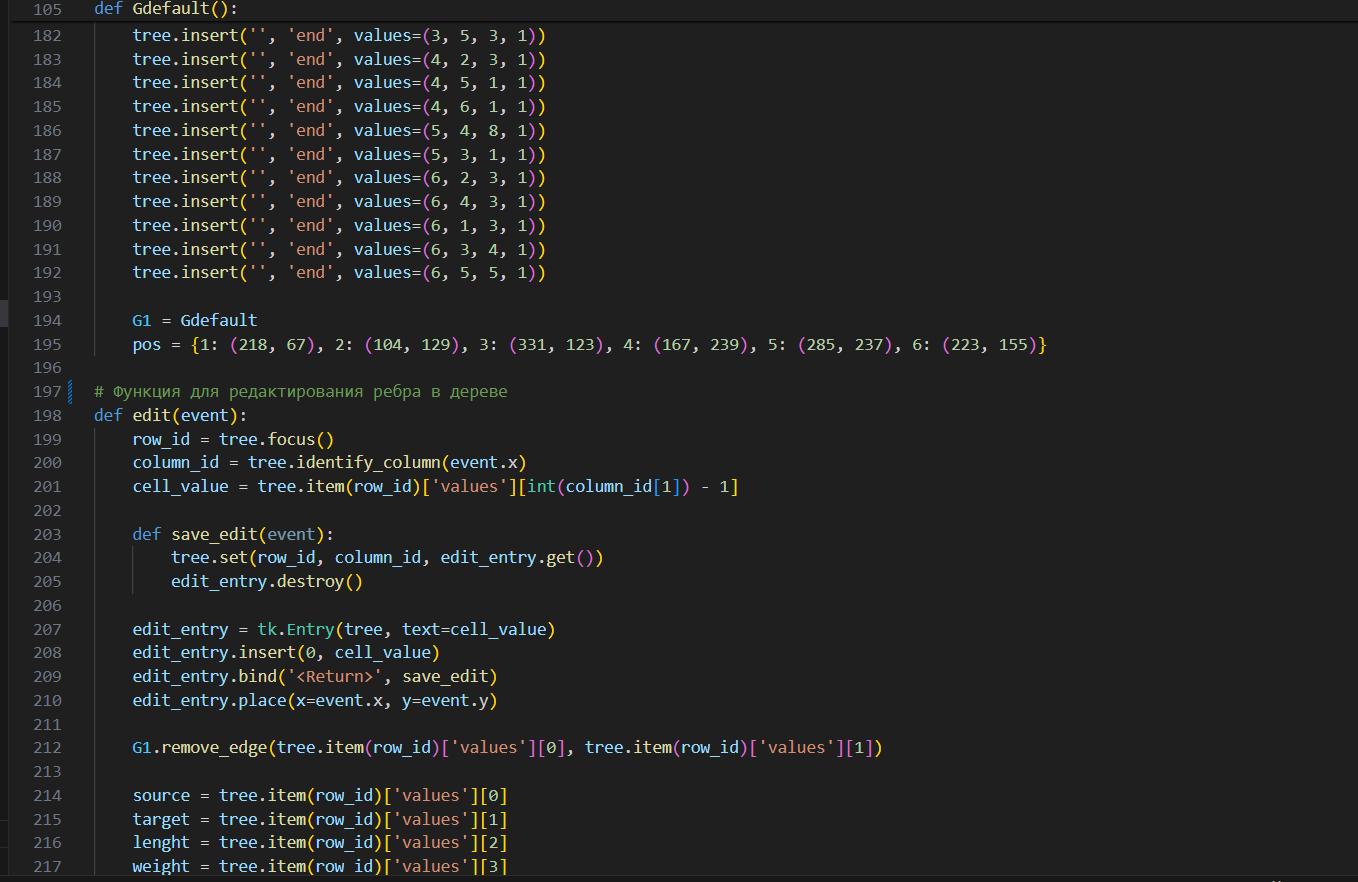
*Рис 4: листинг*



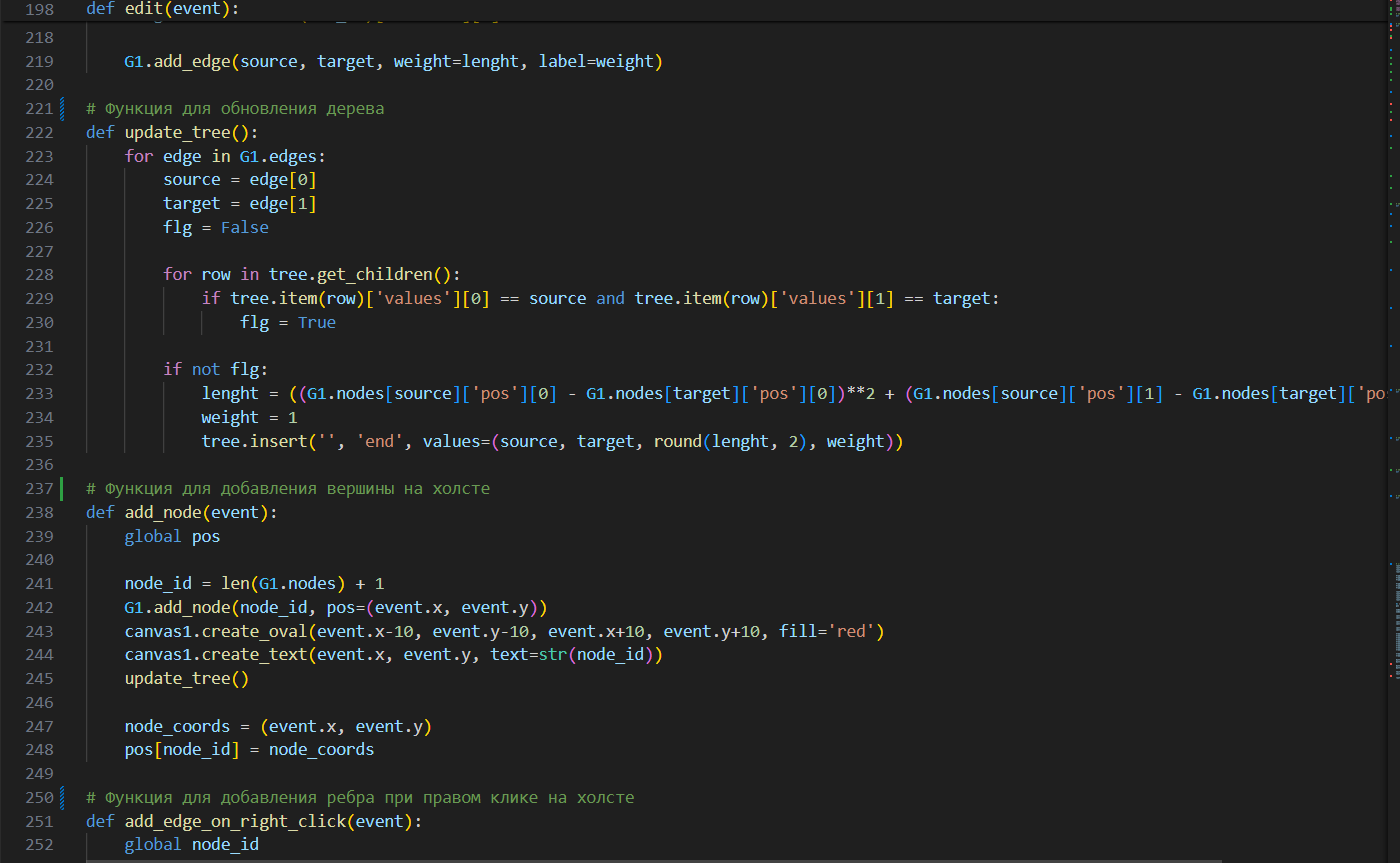
*Рис 5: листинг*



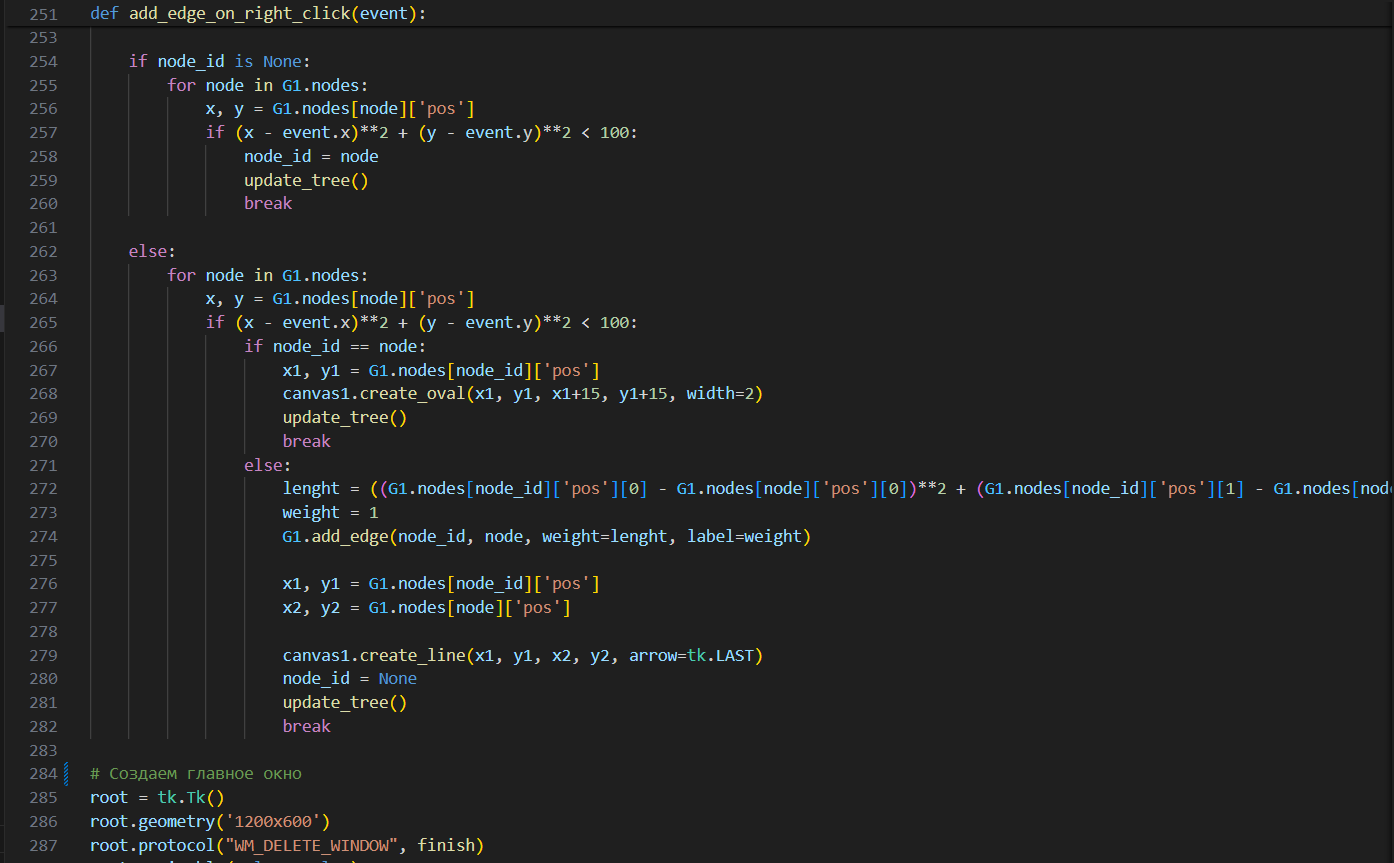
*Рис 6: листинг*



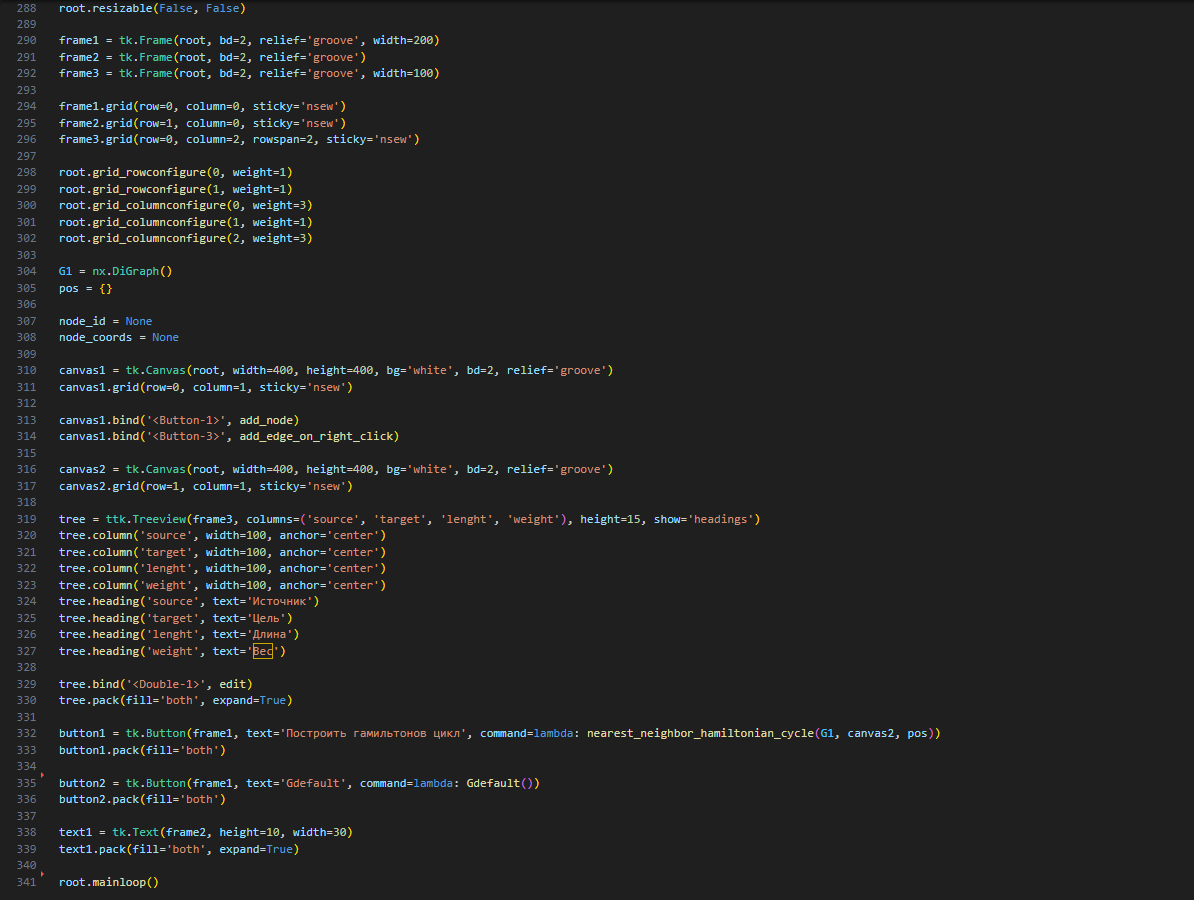
*Рис 7: листинг*



*Рис 8: листинг*



*Рис 9: листинг*



*Рис 10: листинг*