ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»
(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)**

Кафедра анализа данных и машинного обучения
Факультета информационных технологий и анализа больших данных

***Дисциплина: «*Машинное обучение в семантическом и сетевом анализе»**
Направление подготовки: «Прикладная математика и информатика»
Профиль: «Прикладное машинное обучение»
Факультет информационных технологий и анализа больших данных
Форма обучения очная
Учебный 2023/2024 год, 6 семестр

Курсовая работа на тему:
«Реализация интерактивных методов визуализации графов»

Выполнил(а):

студент(ка) группы ПМ21-1

Чеботарева А. В.

Научный руководитель:

доцент к.ф.-м.н. Каверина В. К.

Москва 2024

Оглавление

Введение

Графы — это математическая модель, которая используется для представления сложных взаимосвязей и структур данных. Они играют ключевую роль во многих областях, таких как социальные сети, транспортные сети, биоинформатика и многие другие.

Визуализация графов является важным инструментом для анализа и наглядного представления таких структур данных. С её помощью можно легко увидеть основные характеристики графа: его структуру, связи между вершинами и группировку вершин.

Однако, традиционные методы визуализации графов часто не обеспечивают достаточной интерактивности и возможностей для детального анализа. Поэтому было решено использовать интерактивные методы визуализации, которые позволяют пользователям взаимодействовать с графом, изменять его вид, фильтровать данные, исследовать связи и многое другое.

Целью данной работы: разработка и реализация интерактивных методов визуализации графов на языке программирования Python.

Задачи:

1. Изучить основные понятия теории графов
2. Изучить библиотеки и возможности языка программирования Python для работы с графами
3. Выполнить практическую реализацию интерактивных методов визуализации графов

Таким образом, данная тема является актуальной в современном мире, где все больше данных представлено в виде графов, и эффективная визуализация и анализ таких данных становится все более важным инструментом для исследователей и разработчиков.

Теория графов

Для реализации методов визуализации графов, важно понимать основные понятия теории графов. Давайте рассмотрим ключевые элементы и понятия этой теории, которые помогут лучше разобраться в принципах работы графов и их визуализации.

Основные понятия

Теория графов — это раздел математики, который изучает объекты, называемые графами, состоящими из вершин (узлов) и рёбер (связей), соединяющих эти вершины.

Граф — это математическая абстракция, используемая для представления сетей связей между объектами. Он состоит из двух основных компонентов: вершин (узлов) и ребер (связей между вершинами).

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

Ребро графа — это связь между двумя вершинами, представленная в виде пары вершин. Оно может иметь направление (ориентированное ребро), указывая на однонаправленную связь, или быть без направления (неориентированное ребро), что означает двунаправленную связь.

Инцидентность - вершина и ребро называются инцидентными, если вершина является для этого ребра концевой. Обратите внимание, что термин “инцидентность” применим только к вершине и ребру.

Смежность вершин - две вершины называются смежными, если они инцидентны одному ребру.

Смежность рёбер - два ребра называются смежными, если они инцидентны одной вершине.

Ориентированный граф — это граф, в котором ребра имеют направление, указывающее на однонаправленную связь между вершинами.

Неориентированный граф — это граф, в котором ребра не имеют направления, показывая двунаправленные связи между вершинами.

Подграф — это граф, содержащий некоторые из вершин и ребер исходного графа, при этом все ребра и вершины подграфа принадлежат исходному графу.

Атрибуты узлов — это дополнительная информация, ассоциированная с каждой вершиной графа. Они могут включать в себя любые данные, которые могут быть полезны для анализа или визуализации графа. Например, это могут быть числовые значения, категориальные признаки или текстовые описания вершин.

Эти основные понятия теории графов, включая атрибуты узлов, обеспечивают базовую абстракцию для анализа и визуализации сложных систем и сетей.

Описание библиотек и инструментов Python для визуализации графов

В этой главе мы познакомимся с основными библиотеками и инструментами Python для работы с графами. Эти инструменты позволяют легко отображать графы, исследовать их структуру и анализировать данные. Мы рассмотрим их основные возможности и преимущества.

Jupyter Notebook — это интерактивная среда, которая позволяет создавать и делиться документами, содержащими код, текст, графику и другие элементы. Она позволяет удобно выполнять исследовательский анализ данных, создавать прототипы программ и обмениваться результатами работы с коллегами.

Основным достоинством Jupyter Notebook является то, что каждая ячейка представляет из себя отдельную программу, которая способна исполниться, как независимо от других ячеек, так и взять переменные из уже выполненных ячеек. Данное достоинство позволило стать юпитеру самым удобным инструментом для анализа данных.

1. Ipywidgets

Библиотека ipywidgets (Interactive Python Widgets) представляет собой инструмент для создания интерактивных виджетов в Jupyter notebook и веб-приложениях, используя язык программирования Python. Она позволяет легко добавлять интерактивные элементы управления визуализациям, таким как графам, чтобы пользователи могли взаимодействовать с данными и параметрами.

Преимущества использования ipywidgets включают:

1. Интерактивность: ipywidgets позволяет создавать интерактивные элементы управления, такие как ползунки, кнопки, текстовые поля и другие, что

обеспечивает возможность пользователю манипулировать данными и параметрами в реальном времени.

2. Простота в использовании: ipywidgets предлагает простой и интуитивно понятный интерфейс для создания и настройки интерактивных элементов управления без необходимости в глубоких знаниях веб-разработки.

3. Поддержка различных виджетов: библиотека имеет широкий набор виджетов для создания различных элементов управления, что позволяет настраивать внешний вид и функционал виджетов для лучшего соответствия задаче.

4. Встроенная поддержка Jupyter notebook: ipywidgets интегрирована напрямую с Jupyter notebook, что обеспечивает возможность создавать интерактивные элементы управления прямо в ячейках ноутбука.

2. PyVis

Библиотека PyVis (Python Visualization library) является инструментом для создания интерактивных визуализаций графов в Jupyter notebook и веб-приложениях с использованием языка программирования Python. Она предоставляет простой способ визуализации графов с возможностью интерактивного взаимодействия, такого как перетаскивание узлов, масштабирование, отображение дополнительной информации при наведении курсора и других функций.

Основные черты библиотеки pyvis включают:

1. Интерактивная визуализация: pyvis позволяет создавать интерактивные графы с возможностью взаимодействия пользователя, что делает исследование и визуализацию графов более динамичной и удобной.

2. Настройка внешнего вида: библиотека предоставляет широкие возможности для настройки внешнего вида графов: цвета, размеры узлов и рёбер, стили текста и многое другое.

3. Поддержка различных типов графов: pyvis поддерживает создание различных типов графов, включая ориентированные и неориентированные графы, графы с взвешенными рёбрами и другие.

4. Возможность сохранения в различных форматах: после создания визуализации графа с помощью pyvis, можно сохранить её в различных форматах, таких как изображение PNG, SVG или HTML файл, для дальнейшего использования.

5. Интеграция с Jupyter notebook: Библиотека легко интегрируется с Jupyter notebook, позволяя создавать и отображать интерактивные визуализации графов непосредственно в ячейках ноутбука.

3. NetworkX

NetworkX — это библиотека Python для анализа и создания сложных сетей, включая графы, диаграммы, узловые и краевые списки, а также многие другие структуры данных, используемые для представления и изучения сетевых структур. NetworkX предоставляет широкий спектр функций для генерации, анализа, визуализации и манипулирования сетями, что делает её ценным инструментом для исследования сложных сетей и графов.

NetworkX является одной из самых популярных библиотек для работы с графами.

Основные возможности и функции NetworkX:

1. Поддержка различных типов графов: NetworkX поддерживает создание и работу с различными типами графов, включая ориентированные и неориентированные графы, мультиграфы (графы с параллельными рёбрами), графы с взвешенными рёбрами и многое другое.

2. Генерация и манипуляции графами: библиотека предоставляет множество встроенных методов для генерации различных типов графов, добавления и удаления узлов и рёбер, а также манипуляции структурами графов.

3. Алгоритмы и анализ графов: NetworkX включает большое количество алгоритмов для работы с графами, таких как поиск кратчайшего пути, поиск связных компонент, центральности узлов, поиск циклов и др. Эти алгоритмы позволяют проводить различные анализы структуры и свойств графа.

4. Визуализация графов: библиотека предоставляет возможность визуализации графов с помощью стандартных инструментов или интеграции с внешними библиотеками визуализации, такими как matplotlib или pyvis.

4. IPython.display

Модуль IPython.display позволяет отображать HTML контент в Jupyter notebook. Это полезно при визуализации графов, так как можно встраивать интерактивные веб-графики и диаграммы прямо в ячейках ноутбука. Модуль IPython.display предоставляет простой способ интеграции и отображения HTML/JavaScript содержимого.

Практическая реализация

Изучив теорию графов и познакомившись с библиотеками Python для их визуализации, мы можем перейти к практической реализации задачи. На этом этапе мы применим полученные знания на практике, создавая и визуализируя графы с помощью инструментов Python.

Создание графа:

Пользователь может задать граф для визуализации, используя уже существующий граф из библиотеки NetworkX. Реализованная функция принимает этот граф как входные данные и отображает его в интерактивном режиме.

Интерактивные элементы управления (рисунок 1):

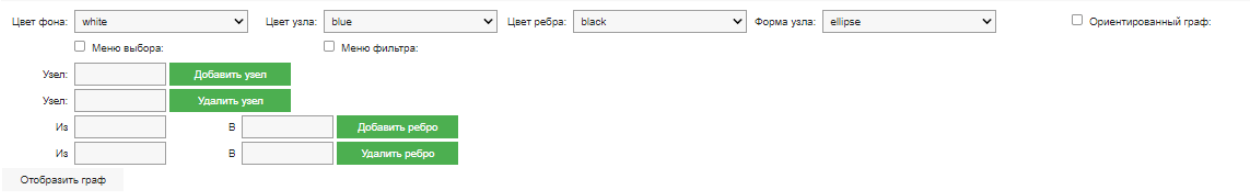

Рисунок 1

Выпадающие списки:

- Выбор цвета фона графа (рисунок 2).
- Выбор цвета узлов (рисунок 3).
- Выбор цвета ребер (рисунок 4).
- Выбор формы узлов (рисунок 5)

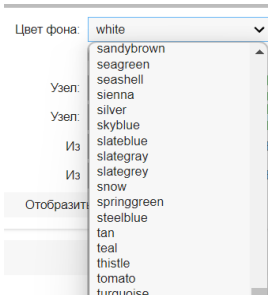

Рисунок 2

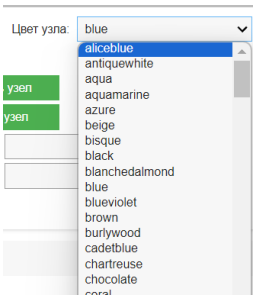

Рисунок 3

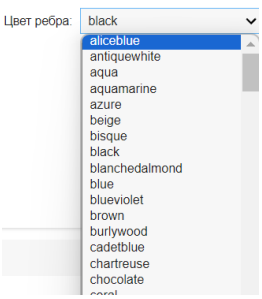

Рисунок 4

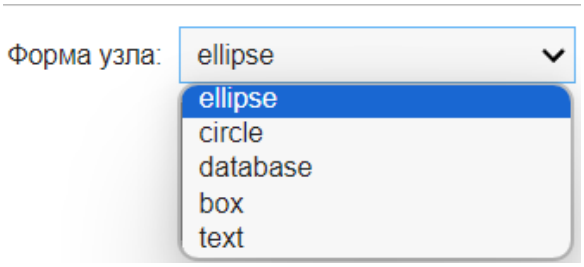

Рисунок 5

Эти списки позволяют выбрать цвета для фона графа, узлов и ребер, а также форму узлов. Пользователь может выбрать цвет из списка 148 предустановленных вариантов в палитре CSS4. Для выбора формы доступны 5 вариантов: эллипс, круг, база данных, бокс, текст. (рисунок 6)

Рисунок 6

Кнопки:

- Кнопка "Добавить узел": добавляет новый узел с заданной меткой.
- Кнопка "Удалить узел": удаляет узел с заданной меткой.
- Кнопка "Добавить ребро": создает ребро между двумя узлами с заданными метками.
- Кнопка "Удалить ребро": удаляет ребро между двумя заданными узлами.
- Кнопка "Отобразить граф": отображает граф с выбранными настройками.

Текстовые поля:

- Поле для ввода метки нового узла.
- Поле для ввода метки удаляемого узла.
- Два поля для ввода меток узлов при добавлении/удалении ребра.

При добавлении/удалении узлов и ребер проводится ряд проверок на корректность ввода, наличие/отсутствие данных в графе. Пользователь информируется о недопустимости того или иного действия. При удалении ребра инцидентные ему вершины остаются в графе. При удалении узла также удаляются все инцидентные ему ребра.

Флажки:

Флажок "Ориентированный граф": переключает между отображением ориентированного и неориентированного графа. При выборе ориентированного графа на ребрах отображаются соответствующие стрелочки (рисунки 7–8).

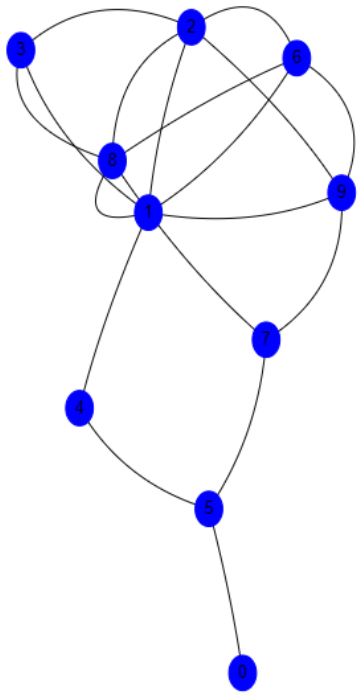

Рисунок 7

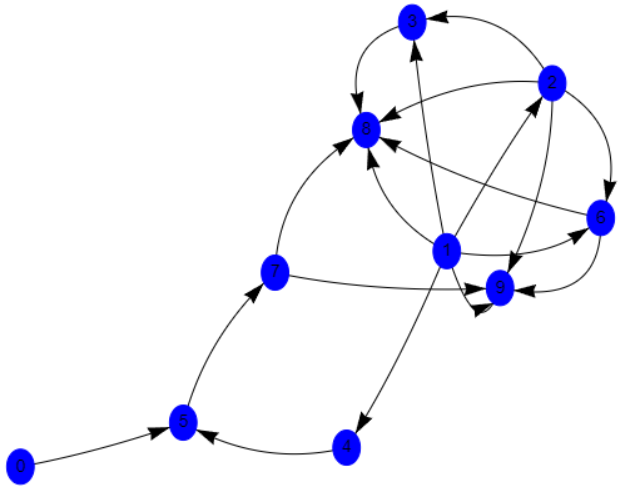

Рисунок 8

Флажок "Меню фильтра": при отображении графа включает меню фильтра.

Когда пользователь активирует опцию "Меню фильтра", на графе появляется дополнительное интерактивное меню с различными критериями фильтрации. Например, пользователь может фильтровать узлы по определенным

атрибутам (например, по id, форме или другим свойствам), либо фильтровать ребра по их атрибутам (например, по id, направлению или другим характеристикам).

После выбора критериев фильтрации пользователь может применить фильтр, в результате чего будет отображаться подграф, узлы и/или ребра которого соответствуют выбранным критериям (рисунок 9). Это позволяет сфокусироваться на определенных частях графа или исследовать его структуру с учетом определенных параметров или свойств.

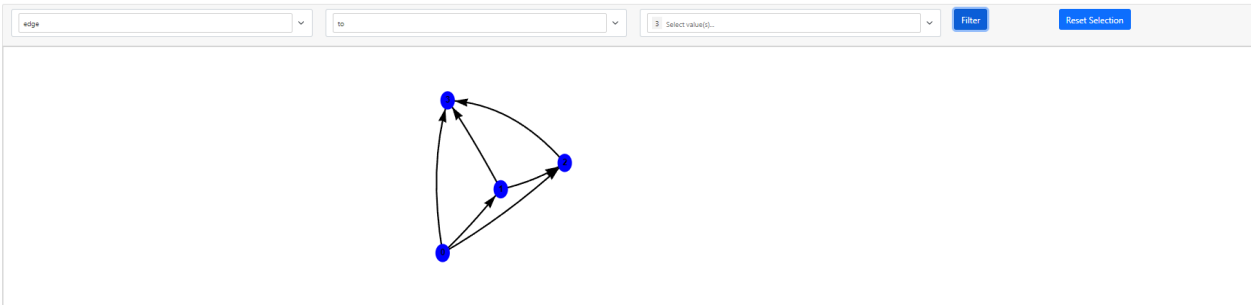

Рисунок 9

Флажок "Меню выбора": при отображении графа включает меню выбора.

При выборе узла подсвечивается выбранный узел, а также инцидентные ему ребра и смежные с ним узлы (рисунок 10). С помощью этой функции можно детально исследовать каждый узел в отдельности.

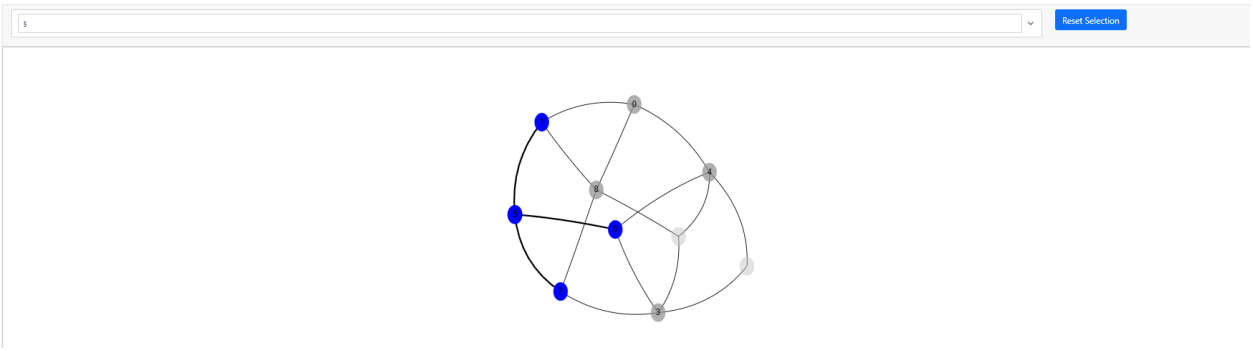

Рисунок 10

Интерактивности визуализации также добавляют встроенные возможности библиотеки pyvis, а именно:

- Менять масштаб графа колесиком мышки
- Перемещать граф
- Двигать вершины
- Отображать атрибуты узлов, при наведении курсора (рисунок 11)

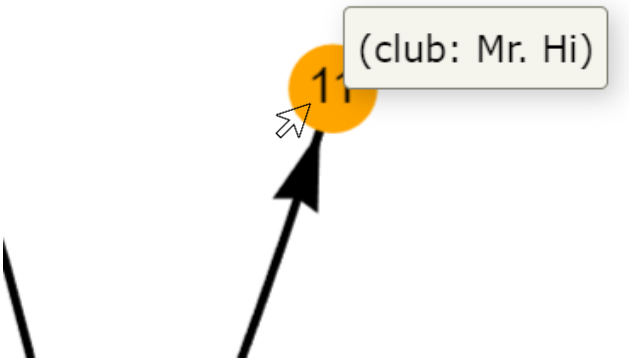

Рисунок 11

Полный рабочий интерфейс представлена на рисунке 12.

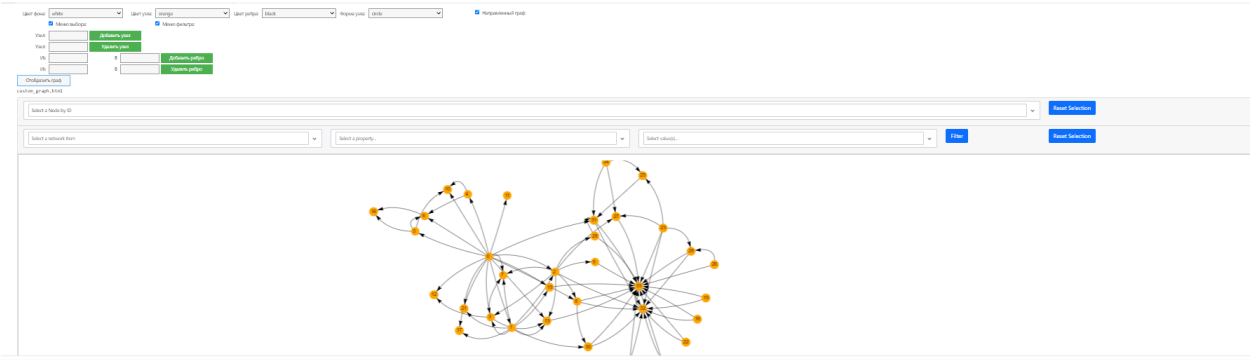

Рисунок 12

Таким образом, используя эти элементы управления, пользователь может:

- Менять внешний вид графа: цвет фона, узлов, ребер, форму узлов.
- Добавлять новые узлы и ребра.

- Удалять существующие узлы и ребра.
- Переключаться между ориентированным и неориентированным представлением графа.
- Использовать меню выбора узлов, чтобы выделить определенные узлы и получить о них дополнительную информацию.
- Использовать меню фильтрации, чтобы отобразить только подмножество графа, соответствующее заданным критериям.

Разработанную функцию визуализации графов можно использовать в качестве модуля, предварительно установив библиотеку pyvis.

Заключение

В рамках данной работы были изучены основные понятия теории графов, возможности библиотек и инструментов языка программирования Python и разработана система интерактивной визуализации графов. Были успешно интегрированы библиотеки NetworkX, PyVis и Ipywidgets, что позволило создать гибкий и удобный инструмент для анализа и исследования графовых данных.

Разработанная система предоставляет пользователю широкий набор интерактивных элементов управления, позволяющих:

- Легко настраивать внешний вид графа, изменяя цвет фона, узлов, ребер, а также форму узлов.
- Динамически добавлять и удалять узлы и ребра, взаимодействуя с графом в реальном времени.
- Переключаться между ориентированным и неориентированным представлением графа.
- Использовать меню фильтрации для выделения определенных подмножеств графа, основываясь на заданных критериях.
- Применять меню выбора для фокусировки на отдельных узлах и изучения их свойств.
- Использовать встроенные возможности библиотеки PyVis для изменения масштаба, перемещения графа и взаимодействия с его элементами.

Данная реализация демонстрирует эффективный подход к интерактивной визуализации графов, делая анализ графовых данных более доступным и наглядным.

В дальнейшем разработанный инструмент может быть расширен дополнительными функциями, такими как:

- Возможность анализировать и отображать дополнительные метрики графа, такие как центральность, кластеризация, связность.

- Возможность изменения расположения вершин графа
- Возможность менять цвета каждой из вершин по отдельности
- Визуальная реализация алгоритмов работы с графами (поиск кратчайшего пути и другие)

Таким образом, данная работа позволила реализовать инструмент для визуализации графов с интерактивными методами, что может быть полезным для исследователей, разработчиков и других специалистов, работающих с графовыми структурами.

Список литературы

1. Документация библиотеки NetworkX
2. Документация библиотеки PyVis
3. Документация библиотеки Ipywidgets
4. Т. И. Бояринцева, А.А. Мастихина. Теория графов. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2014

Приложение

Приложение 1

https://colab.research.google.com/drive/1FQj_OPY5HfCQEypQrBJAZSqTLJ_ahYQN?usp=sharing

Приложение 2

Листинг кода

```
pip install pyvis
def visualize_custom_graph(graph):
    import ipywidgets as widgets
    from pyvis import network as net
    from IPython.display import display, HTML
    import networkx as nx
    import matplotlib.colors as mcolors



    # Создаем новый граф с метками узлов в виде строк
    string_graph = nx.Graph()
    for node, attr in graph.nodes(data=True):
        string_graph.add_node(str(node), **attr)
    for edge in graph.edges():
        string_graph.add_edge(str(edge[0]), str(edge[1]))

    def visualize_graph(bg_color='#FFFFFF', node_color='#1f77b4',
edge_color='black', node_shape='ellipse', directed=False, select_menu=False, fil-
ter_menu=False):
        # Создаем граф
        g = net.Network(height='1000px', width='100%', bgcolor=bg_color,
font_color="black", notebook=True, directed=directed, cdn_resources='remote', se-
lect_menu=select_menu,filter_menu=filter_menu )

        # Добавляем узлы
        for node, attr in string_graph.nodes(data=True):
            # Преобразуем узел в строковое представление
            node_str = str(node)
            # Строим метку, включая атрибуты
            label = f"({', '.join(f'{key}: {value}' for key, value in attr.items())})"
            g.add_node(node_str, label=node_str, title=label, color=node_color, size=20,
shape=node_shape)

        # Добавляем ребра
        for edge in string_graph.edges():
```

```
            g.add_edge(str(edge[0]), str(edge[1]), color=edge_color)

        # Показываем граф
        file_name = 'custom_graph.html'
        g.show(file_name)
        display(HTML(file_name))

    # Получаем список названий цветов из matplotlib
    color_names = list(mcolors.CSS4_COLORS.keys())

    # Создаем виджеты для выбора параметров
    bg_color_dropdown = widgets.Dropdown(options=color_names, value='white', description='Цвет фона:')
    node_color_dropdown = widgets.Dropdown(options=color_names, value='blue', description='Цвет узла:')
    edge_color_dropdown = widgets.Dropdown(options=color_names, value='black', description='Цвет ребра:')
    node_shape_dropdown = widgets.Dropdown(options=['ellipse', 'circle', 'database', 'box', 'text'], value='ellipse', description='Форма узла:')
    directed_checkbox = widgets.Checkbox(value=False, description='Направленный граф:')
    select_menu_checkbox = widgets.Checkbox(value=False, description='Меню выбора:')
    filter_menu_checkbox = widgets.Checkbox(value=False, description='Меню фильтра:')
    add_node_text = widgets.Text(description='Узел:', layout=widgets.Layout(width='200px'))
    remove_node_text = widgets.Text(description='Узел:', layout=widgets.Layout(width='200px'))
    add_edge_from_text = widgets.Text(description='Из', layout=widgets.Layout(width='200px'))
    add_edge_to_text = widgets.Text(description='В', layout=widgets.Layout(width='200px'))
    remove_edge_from_text = widgets.Text(description='Из', layout=widgets.Layout(width='200px'))
    remove_edge_to_text = widgets.Text(description='В', layout=widgets.Layout(width='200px'))
    add_node_button = widgets.Button(description='Добавить узел', button_style='success')
    remove_node_button = widgets.Button(description='Удалить узел', button_style='success')
    add_edge_button = widgets.Button(description='Добавить ребро', button_style='success')
```

```
    remove_edge_button = widgets.Button(description='Удалить ребро', but-
ton_style='success')
    generate_button = widgets.Button(description='Отобразить граф')

    # Обработчик события для кнопки добавления узла
    def on_add_node_button_clicked(b):
        with output:
            output.clear_output()
            new_node_label = add_node_text.value.strip()  # Получаем введенный
текст
            if new_node_label != '':
                if new_node_label not in string_graph.nodes():
                    string_graph.add_node(new_node_label)
                else:
                    print("Узел уже существует!")
            else:
                print("Пожалуйста, введите метку узла.")
            visualize_graph(bg_color_dropdown.value, node_color_dropdown.value,
edge_color_dropdown.value, node_shape_dropdown.value, directed_checkbox.value,
select_menu_checkbox.value, filter_menu_checkbox.value)

    # Обработчик события для кнопки удаления узла
    def on_remove_node_button_clicked(b):
        with output:
            output.clear_output()
            node_to_remove = remove_node_text.value.strip()
            # Получаем введенный текст
            if node_to_remove != '':
                if node_to_remove in string_graph.nodes():
                    string_graph.remove_node(node_to_remove)
                else:
                    print("Узел не найден!")
            else:
                print("Пожалуйста, введите метку узла.")
            visualize_graph(bg_color_dropdown.value, node_color_dropdown.value,
edge_color_dropdown.value, node_shape_dropdown.value, directed_checkbox.value,
select_menu_checkbox.value, filter_menu_checkbox.value)

    # Обработчик события для кнопки добавления ребра
    def on_add_edge_button_clicked(b):
        with output:
            output.clear_output()
            edge_from = add_edge_from_text.value.strip()
            edge_to = add_edge_to_text.value.strip()
```

```
            # Получаем введенный текст
            if edge_from != '' and edge_to != '':
                if edge_from in string_graph.nodes() and edge_to in string_graph.nodes():
                    string_graph.add_edge(edge_from, edge_to)
                else:
                    print("Один или оба узла не найдены!")
            else:
                print("Пожалуйста, введите исходный и целевой узлы.")
            visualize_graph(bg_color_dropdown.value, node_color_dropdown.value,
edge_color_dropdown.value, node_shape_dropdown.value, directed_checkbox.value,
select_menu_checkbox.value, filter_menu_checkbox.value)

    # Обработчик события для кнопки удаления ребра
    def on_remove_edge_button_clicked(b):
        with output:
            output.clear_output()
            edge_from = remove_edge_from_text.value.strip()
            edge_to = remove_edge_to_text.value.strip()
            # Получаем введенный текст
            if edge_from != '' and edge_to != '':
                if string_graph.has_edge(edge_from, edge_to):
                    string_graph.remove_edge(edge_from, edge_to)
                else:
                    print("Ребро не найдено!")
            else:
                print("Пожалуйста, введите исходный и целевой узлы.")
            visualize_graph(bg_color_dropdown.value, node_color_dropdown.value,
edge_color_dropdown.value, node_shape_dropdown.value, directed_checkbox.value,
select_menu_checkbox.value, filter_menu_checkbox.value)

    # Обработчик события для кнопки генерации графа
    def on_generate_button_clicked(b):
        with output:
            output.clear_output()
            visualize_graph(bg_color_dropdown.value, node_color_dropdown.value,
edge_color_dropdown.value, node_shape_dropdown.value, directed_checkbox.value,
select_menu_checkbox.value, filter_menu_checkbox.value)

    # Привязываем обработчики к кнопкам
    add_node_button.on_click(on_add_node_button_clicked)
    remove_node_button.on_click(on_remove_node_button_clicked)
    add_edge_button.on_click(on_add_edge_button_clicked)
    remove_edge_button.on_click(on_remove_edge_button_clicked)
    generate_button.on_click(on_generate_button_clicked)
```

```
    # Создаем блок виджетов и область вывода
    output = widgets.Output()
    display(widgets.VBox([
        widgets.HBox([bg_color_dropdown, node_color_dropdown,
edge_color_dropdown, node_shape_dropdown, directed_checkbox]),
        widgets.HBox([select_menu_checkbox, filter_menu_checkbox]),
        widgets.HBox([add_node_text, add_node_button]),
        widgets.HBox([remove_node_text, remove_node_button]),
        widgets.HBox([add_edge_from_text, add_edge_to_text, add_edge_button]),
        widgets.HBox([remove_edge_from_text, remove_edge_to_text, re-
move_edge_button]),
        generate_button,
        output
    ]))
```