**МИНОБРНАУКИ РОССИИ**



Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Российский государственный гуманитарный университет»**

**(ФГБОУ ВО «РГГУ»)**

|  |
| --- |
| **Отделение интеллектуальных систем в гуманитарной сфере** |
|  |
| **Кафедра математики, логики и интеллектуальных систем в гуманитарной сфере** |
|  |

|  |
| --- |
| **Выпускная квалификационная работа** |
| **бакалаврская работа** |

|  |  |  |  |
| --- | --- | --- | --- |
| **на тему:** | **РАЗРАБОТКА МОДУЛЯ ВВОДА ОТВЕТА ОБУЧАЕМОГО** | | |
| **В ВИДЕ ГРАФА** | | | |
|  | | | |
| **Выполнил обучающийся** | | **Хакимова Муътабар Мансуровна** | |
|  | | | |
| **направление подготовки** | | |
| **45.03.04 Интеллектуальные системы в гуманитарной сфере** | | |
|  | | |
| **направленность (профиль)** | | |
| **Разработка и программирование интеллектуальных систем в гуманитарной сфере** | | |
|  | | |
| **4 курс, очная форма обучения** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Руководитель ВКР** | | |
| **д-р физ.-мат. наук, проф. Аншаков О.М.** |  |  |
|  |  | подпись |
| «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_\_\_ г. | | |

|  |  |  |
| --- | --- | --- |
| **Заведующий кафедрой** | | |
|  | | |
| **д-р физ.-мат. наук, проф. Бениаминов Е.М.** |  |  |
|  |  | подпись |
| «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_\_\_ г. | | |

**Москва 2024 г.**

**СОДЕРЖАНИЕ.**

[**ВВЕДЕНИЕ.** 3](#_Toc169412459)

[**1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.** 5](#_Toc169412460)

[1.1 Основные понятия теории графов. 5](#_Toc169412461)

[1.2 Используемы инструменты для визуализации графов. 12](#_Toc169412462)

[**Dash.** 12](#_Toc169412463)

[**Dash-cytoscape.** 19](#_Toc169412464)

[**2.РАЗРАБОТКА МОДУЛЯ.** 28](#_Toc169412465)

[Архитектура модуля. 28](#_Toc169412466)

[**Язык программирования и IDE.** 28](#_Toc169412467)

[**Реализация модуля.** 29](#_Toc169412468)

[**ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ.** 42](#_Toc169412469)

[**ЗАКЛЮЧЕНИЕ.** 43](#_Toc169412470)

[**ЛИТЕРАТУРА** **& ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ**. 45](#_Toc169412471)

# **ВВЕДЕНИЕ.**

Необходимость создания модуля ввода ответа обучаемого в виде графа обусловлена стремлением к усовершенствованию процесса обучения с применением современных технологий. Такой модуль позволит повысить эффективность образовательного процесса, облегчив взаимодействие преподавателя и студента. При этом использование графов для передачи информации становится все более популярным методом обучения, что делает разработку данного модуля наиважнейшей задачей в контексте современного образования.

Традиционные текстовые и табличные методы представления информации зачастую не в полной мере справляются с задачей наглядного и интуитивно понятного отображения сложных данных. В этом контексте разработка модуля ввода ответов обучаемых в виде графов приобретает особую значимость. Такой модуль не только облегчит процесс обучения, но и позволит значительно повысить его эффективность, предоставив студентам инструмент для активного участия в учебном процессе.

Графы обладают рядом преимуществ, делающих их применение в образовательной среде особенно актуальным:

1. Наглядность: Возможность визуального представления сложных взаимосвязей между элементами данных.
2. Интерактивность: Студенты могут активно взаимодействовать с графами, что способствует лучшему усвоению материала.
3. Универсальность: Графы применимы в различных дисциплинах, таких как математика, информатика, биология и социология.
4. Анализ и сравнение: Возможность сравнения графов позволяет оценивать правильность выполнения заданий и анализировать ошибки.

Цель данной работы заключается в разработке модуля для ввода ответов обучаемых в виде графов, который обеспечит следующие функциональные возможности:

* Создание и редактирование графов с использованием интуитивно понятного интерфейса.
* Сохранение созданных графов в формате JSON для последующего анализа и обработки.
* Ввод ответа обучаемого.

Для реализации модуля будут использованы современные технологии и инструменты, такие как библиотеки Dash и его компоненты, Dash-Cytoscape, которые предоставляют эффективные средства для визуализации графов и создания интерактивных пользовательских интерфейсов.

Таким образом, создание такого модуля отвечает потребностям современных образовательных систем и способствует улучшению качества образования в целом. Дальнейшие разделы работы будут посвящены теоретическим основам теории графов, используемым инструментам для их визуализации, процессу разработки модуля и его практическому применению в образовательном процессе.

# **1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.**

## 1.1 Основные понятия теории графов.

Теория графов — это раздел математики, изучающий структуры, состоящие из узлов (вершин) и соединяющих их линий (рёбер), которые моделируют разнообразные системы и явления.

Графы являются одной из фундаментальных структур данных, широко используемых в математике и информатике для моделирования разнообразных систем и процессов, от социальных сетей до сетей связи и биологических систем. Формально граф (G) определяется как пара множеств (V, E), где V – множество вершин (узлов), а E – множество ребер, соединяющих пары вершин.

Вершина (или узел) — это элемент графа, который представляет собой точку схождения или выхода рёбер.

Ребром – это связь между парой смежных вершин в графе.

Подграф – граф, состоящий из подмножества вершин и ребер исходного графа.

Основные понятия графов:

* Графы является направленными (ориентированный), если рёбра имеют направление, или ненаправленными, если рёбра не имеют направления.
* Графы является связными. Граф является таковым, если существует связь между любыми двумя его узлами, в ином случае, граф называется несвязным.
* Графы могут быть цикличными. Цикл в графе представляет собой путь, который начинается и заканчивается в одной и той же вершине без повторения рёбер. Если циклы в графе отсутствуют, такой тип называется Ацикличным. Свойство ацикличности является важным для таких структур как деревья и направленных ациклических графов (DAG).
* Мультиграф – это такой вид графа, в котором между одной и той же парой вершин может существовать несколько ребер. Такой вид связи необходим, например, если есть несколько вариантов путей соединения.
* Взвешенный граф. В таком графе к каждому ребру присваивается значение, которое может представлять собой стоимость, расстояние, время или любая друга мера.
* Графы с петлей – это такой вид графов, в котором вершина соединяется с самим собой.
* Смежность — это такой вид связи, при котором два ребра соединены одной вершиной.
* Простой граф — это граф, в котором между любыми парой вершин существует не более одного ребра, и отсутствуют петли. Простой граф является наиболее распространённым типом графов и используется в различных моделях и задачах.
* Псевдограф — это такой вид графа, в котором есть и петли, и кратные ребра.
* Смешанный граф – граф, который имеет как ориентированные, так и неориентированные ребра.
* Дерево — это особый вид связного ациклического графа, в котором существует единственный путь между любыми двумя вершинами. Деревья широко используются для представления иерархических структур, таких как организационные схемы, файловые системы или эволюционные деревья.
* Планарный граф – это вид графов, который строится таким образом, чтобы его ребра не пересекались.
* Цикл Граф – это граф, в котором вершины соединены в замкнутую цепочку. Циклы могут быть простыми (без повторения рёбер и вершин, кроме начальной и конечной) и сложными.
* Эйлеров граф – это такой вид графа, в котором можно пересечь ребра равно один раз.
* Гамильтонов граф – это вид графа, в котором существует цикл, который обходит все его вершины ровно один раз, при этом исключая возможность прохождения через любую вершину более одного раза.
* Пустой граф – граф G(V,E), который не содержит рёбер, то есть E=∅. В этом графе вершины не соединены друг с другом​.

Деревья.

Деревья являются особым видом графов и представляют собой важный объект изучения в теории графов. Они широко используются в различных областях информатики и математики для моделирования иерархий, поиска и сортировки данных, анализа структуры и принятия решений. Основные свойства деревьев делают их незаменимыми при решении множества практических задач.

Дерево — это особый вид графа, который является связным и ациклическим. Формально дерево (T) — это связный ациклический граф. Так как в дереве существует единственный путь между любыми двумя вершинами, это делает его отличным выбором для отображения иерархических структур и связей между родителем и потомком. Деревья широко используются для моделирования иерархических структур, таких как файловые системы, организационные схемы и эволюционные деревья.

Теорема о количестве ребер гласит, что если в дереве p-вершин, то ребер q = p – 1.

Структура дерева:

* Связность. Между любой парой вершин в дереве существует ровно один путь.
* Ацикличность. В дереве отсутствуют циклы.
* Корень дерева. Специальная вершина, которая может быть выбрана в качестве начальной точки для ориентированного дерева.
* Лист (листовая вершина). Вершина, которая не имеет потомков.
* Внутренние вершины. Вершины, имеющие потомков, то есть не являющиеся листьями.

Типы деревьев:

* Бинарное дерево: Каждая вершина имеет не более двух потомков.

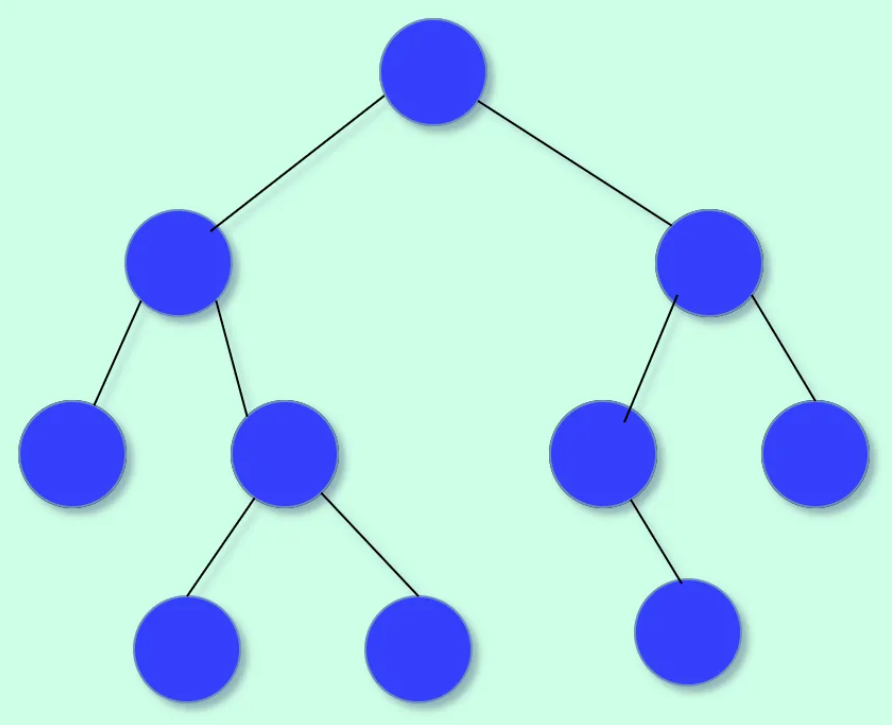


Рис.1 Пример Бинарного дерева

* Поисковое бинарное дерево: Бинарное дерево, в котором левый потомок каждой вершины содержит значения, меньшие, чем значение в самой вершине, а правый – значения большие.

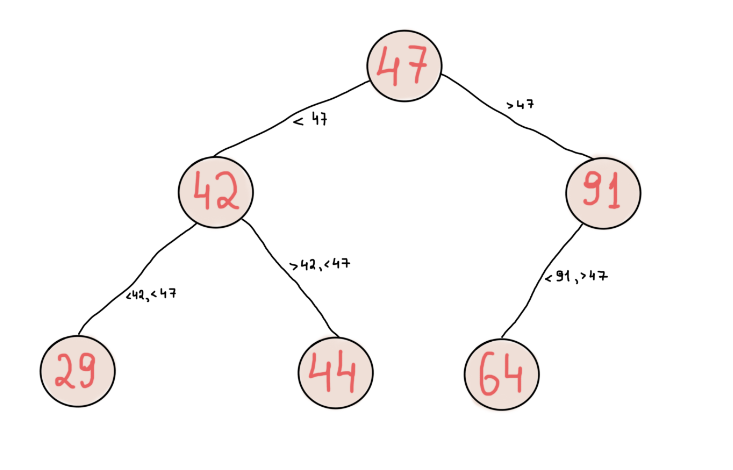


Рис.2 Пример поискового бинарного дерева

* N-арное дерево: Каждая вершина может иметь до N потомков.
* Дерево решений: используется для моделирования процессов принятия решений.

Ранее мной было упомянуто, что целью создания модуля является предоставление помощи при обучении. Может возникнуть логический вопрос для чего же стоит тратить свое время на изучение теории графов, в частности деревьев. То есть значимость изучения графовых деревьев для обучаемых.

Деревья обеспечивают высокую производительность при выполнении операций поиска, добавления и удаления данных благодаря своей иерархической организации, что позволяет сократить количество шагов для достижения желаемого результата. В частности, в сбалансированных структурах данных, таких как AVL-деревья и красно-черные деревья, высота дерева ограничивается логарифмическим значением количества элементов, что обеспечивает быстроту обработки данных.

Деревья идеально подходят для создания моделей иерархических структур, что делает их незаменимыми в таких областях, как анализ данных, разработка алгоритмов и проектирование систем. Они позволяют четко организовать информацию и установить отношения между различными элементами, что облегчает понимание и анализ сложных систем.

Одним из ключевых преимуществ деревьев является их адаптивность к изменениям в объеме данных. Благодаря механизмам балансировки, деревья могут автоматически адаптироваться к росту или уменьшению количества данных, сохраняя при этом высокую производительность операций. Это делает их идеальным выбором для систем, требующих масштабируемости и гибкости.

Понимание принципов работы деревьев является фундаментом для разработки и оптимизации сложных алгоритмов и структур данных. Многие продвинутые алгоритмы, такие как алгоритмы сортировки, поиска и машинного обучения, основаны на использовании деревьев. Изучение деревьев дает студентам необходимые знания и навыки для дальнейшего исследования и разработки в этих областях.

Работа с деревьями способствует развитию логического мышления и аналитических навыков. Студенты учатся анализировать структуры данных, оптимизировать процессы и принимать обоснованные решения, что является ценным навыком в любой профессиональной деятельности.

Далее разберем способы представления деревьев в компьютерных системах, их существует несколько:

Списки смежности: Каждая вершина хранит ссылки на своих потомков.

Матрица смежности: Квадратная матрица, где элементы указывают на наличие рёбер между вершинами.

Кодирование Прюфера: Способ представления деревьев с помощью последовательностей.

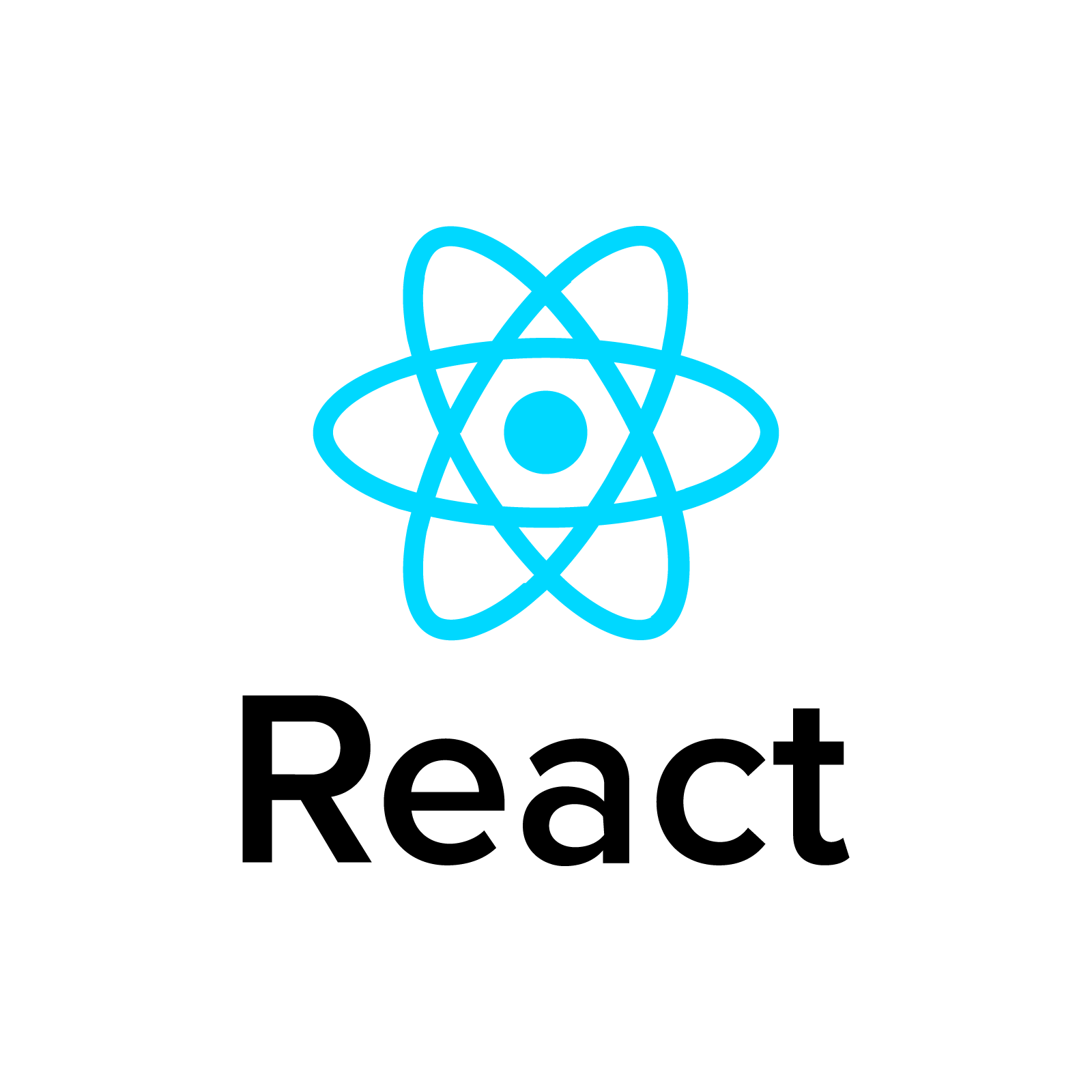
Для работы с деревьями существуют такие алгоитмы:

* Обходы дерева: Прямой (preorder), центрированный (inorder) и обратный (postorder) обходы.
* Поиск в дереве: Алгоритмы поиска в глубину (DFS) и поиска в ширину (BFS) используются для нахождения элементов в дереве.
* Построение дерева: Алгоритмы для построения деревьев из заданных данных, таких как список рёбер или последовательности вершин.

Деревья являются важной структурой данных, которая используется для моделирования иерархий, оптимизации процессов и анализа структур. Их свойства и различные типы позволяют эффективно решать множество задач в информатике, биологии, лингвистике и других областях. Разработка алгоритмов для работы с деревьями и их сравнением предоставляет широкие возможности для анализа и оптимизации данных.

## 1.2 Используемы инструменты для визуализации графов.

### **Dash.**





**+** **+**

**=**

Рис.3 Логотип Dash и составляющие его компоненты.

Хотя данное утверждение не является строго обязательным, важно обладать знаниями какие основные компоненты были использованы для создания фреймворка Dash и их зависимостей.

Dash — мощный и удобный фреймворк, который был выпущен в 2017 году как библиотека Python, это универсальный инструмент для использования в разработке различных типов веб-приложений. Dash может быть помощником для создания, например, аналитических веб-приложений или тех, которые предназначены для визуализации графов и интерактивного взаимодействия.

Преимущества использования Dash:

* Dash позволяет нам создавать полностью интерактивные данные, веб-приложения, аналитику и интерфейсы на чистом Python, не беспокоясь о HTML, CSS и Javascript.
* Dash разработан командой Plotly, что означает отличную интеграцию с библиотекой Plotly. Plotly и, следовательно, Dash являются отличным выбором для создания веб-приложений, потому что эти интерактивные графики сами по себе основаны на веб-технологиях.
* Dash является простым и универсальным в использовании.
* Dash позволяет использовать интерактивные виджеты, такие как ползунки, текстовые поля, кнопки и выпадающие меню. Эти элементы управления позволяют пользователям динамически взаимодействовать с данными, выбирая, какие данные отображать на графиках и диаграммах.
* Dash построен поверх активно используемой системы Flask, что дает нам множество вариантов развертывания, начиная от полностью управляемых до самостоятельно управляемых.
* Dash обеспечивает высокую степень интерактивности через callback- функции, которые позволяют обновлять элементы интерфейса в реальном времени на основе действ пользователя.
* Dash можно расширять с помощью дополнительных библиотек, например dash-cytoscape для визуализации графов.

Основные компоненты Dash:

* Dash: основной компонент, который в свою очередь включает в себя Layout и Callback
* Dash core components (обозначается как dcc): предоставляет набор интерактивных компонентов. В него входят выпадающие списки, датчики выбора даты, слайдеры и многие другие компоненты.
* Dash HTML component: этот компонент предоставляет все доступные HTML теги в виде классов Python.
* Dash Bootstrap components: представляет собой стороннюю библиотеку, которая добавляет функциональность Bootstrap в Dash. Этот пакет и его компоненты заботятся о многих опциях, связанных с макетами и визуальными сигналами. Расположение элементов рядом друг с другом или друг поверх друга, указание их размеров в зависимости от размера экрана браузера и предоставление набора закодированных цветов для более эффективного общения с пользователями являются некоторыми преимуществами использования этой библиотеки.

Инсталляция Dash:

Для того, чтобы начать использование Dash, его надо сначала установить. Это можно сделать таким образом:

Открыть среду разработки, открыть терминал и набрать следующее:



Процесс установки других компонентов dash является аналогичным.

Общая структура Dash.

|  |  |
| --- | --- |
| Части приложения | Пример кода |
| Импорт необходимых данных |  |
| Создание экземпляра Dash |  |
| Определение макета приложения. Макет состоит из компонентов, которые определяют внешний вид приложения (Layout: список HTML и/или интерактивные компоненты). |  |
| Callback функции (Обратные вызовы управляют взаимодействием пользователя с приложением. |  |
| Запуск |  |

1. Импорт необходимых данных.

Импорт библиотек является первым шагом при создании приложения Dash. Он включает в себя загрузку всех необходимых модулей и библиотек, которые будут использоваться в приложении.

1. Создание экземпляра Dash.

Для создания экземпляра приложения используется класс Dash из библиотеки dash. Этот класс инициализирует приложение и позволяет настроить его основные параметры.

«\_name\_» значение для параметра name используется для того, чтобы облегчить для Dash поиск статических активов в приложении.

Так же эта часть приложения позволяет подключить внешний стиль CSS для улучшения внешнего вида приложения, например за счет подключения Bootstrap

1. Layout (макет).

Макет в Dash определяет структуру и организацию компонентов, которые составляют веб-приложение. Он определяет, как и где каждый компонент будет размещен в приложении. Этот макет строится с использованием иерархической структуры "компонентов", которые могут быть простыми HTML элементами или более сложными интерактивными элементами.

Некоторые особенности:

* Строится с использованием дерева компонентов, таких как html.Div и dcc.Graph.
* HTML Компоненты Dash (dash.html): содержит компонент для каждого HTML тега. Эти компоненты используются для создания базовой структуры приложения (Пример: html.H1(children='Привет, Dash')).
* Не все компоненты являются чистыми HTML. Модуль Dash Core Components (dash.dcc) содержит интерактивные компоненты более высокого уровня, создаваемые с помощью JavaScript, HTML и CSS через библиотеку React.js.
* Основные Компоненты: dash.dcc содержит более сложные интерактивные компоненты.
* Декларативный Синтаксис: Компоненты описываются с использованием атрибутов ключевых слов.
* Свойство Children: Специальный атрибут, который может содержать различные типы контента.
* Пользовательские Стили: возможность использовать пользовательский CSS для изменения стандартных стилей элементов.

1. Callback (обратный вызов).

Обратный вызов в Dash позволяет построить интерактивное приложение, которое автоматически вызывает функции при каждом изменении состояния компонента. Это основополагающий характер взаимодействия с пользователем, обеспечивающий возможность для приложения обновлять данные или визуализацию в ответ на любое действие пользователя - щелчок мыши или выбор из раскрывающегося меню.

Основные компоненты callback:

* Входные данные (input): callback принимает входные значения, которые могут быть свойствами компонента - например, значение раскрывающегося объекта (dcc.Dropdown) или положение ползунка (dcc.Slider).
* Функция обратного вызова: эта функция обрабатывает входные данные и возвращает новые значения для применения к выходным (output) компонентам.
* Выходные данные (output): указывает, какой компонент и его свойство будут обновлены в результате функции обратного вызова.

Расширенные возможности callback.

1. Возможность использовать множество входных и выходных данных:

В Dash любой “output” может иметь несколько “input” компонентов.

Обычно функции callback обновляют один “output” свойство, между тем, стоит заметить, что Dash позволяет обновлять сразу несколько свойств output компонентов из одной callback функции, что может быть особенно полезным, если два вывода зависят от одного и того же трудоемкого промежуточного результата, такого как медленный запрос к базе данных.

Функции обратного вызова в Dash могут быть связаны, позволяя создавать более сложные и динамичные пользовательские интерфейсы. В этом случае выход одного обратного вызова может служить входом для другого. Это полезно для создания интерфейсов, где один элемент управления влияет на доступные настройки другого элемента управления.

1. Использование State. State позволяет сохранять состояние компонентов между вызовами callback.

Callback функции в Dash представляют собой мощный и гибкий инструмент для создания интерактивных веб-приложений. Они позволяют реализовать сложные сценарии взаимодействия, обеспечивая высокую производительность и удобство использования. Мной было представлено несколько основных возможностей callback, тем не менее, Dash предоставляет значительно больше расширенных возможностей для реализации самых разнообразных задач.

1. Запуск.

Для запуска приложения используется run\_server, который запускает локальный сервер и делает его доступным в веб-браузере.

(debug=True) включает режим отладки, используется для того, чтобы приложение автоматически перезагружалось.

### **Dash-cytoscape.**

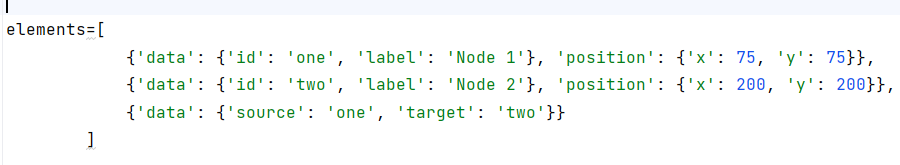
В предыдущем обсуждении при описании особенностей Dash, подчеркивалась возможность расширения функционала Dash за счет использования дополнительных библиотек. В ходе исследований установлено, что одним из наиболее эффективных инструментов для визуализации графов является Dash-Cytoscape. Этот компонент Dash, построенный на основе библиотеки JavaScript Cytoscape.js, использует её основные функции и возможности для создания и анализа сетевых графов. Dash-Cytoscape позволяет создавать легко настраиваемые, высокопроизводительные, интерактивные и веб-базированные сети, обеспечивает глубокую интеграцию с макетами Dash и обратными вызовами, создавая мощную сеть с богатым набором компонентов Dash.

Для того, чтобы начать пользоваться Dash-cytoscape, необходимо установить соответствующий пакет, для этого надо ввести в терминал IDE: pip install dash-cytoscape, затем импортировать в Dash-приложение.

Элементы Dash-cytoscape.

Каждый элемент определяется словарём, в котором указывается его назначение и описываются его свойства. Обычно указывается к какой группе принадлежит элемент (например, если это узел или ребро), далее указывается какая позиция является желаемым для элемента (если это узел) или какие данные он содержит. Фактически, данные и ключи положения сами сопоставляются с словарями, где каждый элемент указывает аспект данных или позиции.

Для узлов обычно указывается идентификатор (id) и позиция (position), а для ребер — идентификаторы начального (source) и конечного (target) узлов.

Рис.4 Элементы графового дерева.

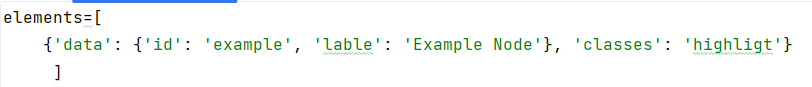
Важно отметить, что элементы так же подлежать стилизации, что позволяет создавать визуально привлекательные и информативные графы.

Таблица стилей (Stylesheet) в Dash Cytoscape позволяет определить внешний вид элементов, используя селекторы, аналогичные тем, что используются в CSS. Селекторы могут быть направлены на узлы (node), ребра (edge) и классы элементов (class). С помощью таблицы стилей можно задать различные свойства, такие как цвет фона (background-color), форма (shape), размер (size), шрифт (font-family), цвет текста (color) и многие другие.



Рис.5 Пример кода со стилизацией элементов.

Стили могут быть применены к отдельным элементам, к группам элементов или ко всему графу. Чтобы применить стили к элементам, необходимо добавить атрибут classes в их определение, а затем использовать эти классы в таблице стилей для указания внешнего вида.



В дополнение к свойствам, представленным выше, словарь элемента может также принимать булевые элементы, которые задают его состояние.

Например, узлы графов можно предотвращать перемещение узла, подсвечивать выбранный узел или наоборот, определять можно ли перетаскивать узел.

Макеты (Cytoscape Layouts).

Эстетическая привлекательность и информативность графа являются результатом не только использования стилизации, но и эффективного расположения. Размещение узлов и рёбер в сетевом графе осуществляется с помощью макетов. Правильный выбор макета значительно улучшает восприятие данных, делая структуру графа более ясной и понятной.

Макеты в Dash Cytoscape предоставляют различные методы размещения элементов графа, каждый из которых подходит для различных типов данных и аналитических задач. Макеты позволяют автоматически размещать узлы и рёбра таким образом, чтобы они занимали оптимальное положение, минимизируя пересечения и улучшая читаемость графа, что пересекается с моими целями разработки модуля, то есть помогает повысить эффективность обучения.

Параметр layout компонента cyto.Cytoscape принимает словарь, который определяет, как узлы должны быть размещены на экране. Каждый граф требует наличие этого словаря с указанным значением ключа name. Это представляет собой встроенный метод отображения, который может быть одним из следующих:

* Preset:

Предварительная компоновка (preset) в Dash-Cytoscape - уникальная и очень полезная компоновка, позволяющая полностью контролировать позиции узлов. При инициализации графов можно указать, что узлы должны быть размещены на заранее определенных позициях. Эти позиции указываются в поле position каждого узла в его JSON-объективной форме. Такая возможность располагать узлы может быть идеальным выбором для случаев, когда узлы должны быть расположены на основе внешних данных, таких как географические координаты, вручную разработанные макеты или другие заранее определенные схемы.

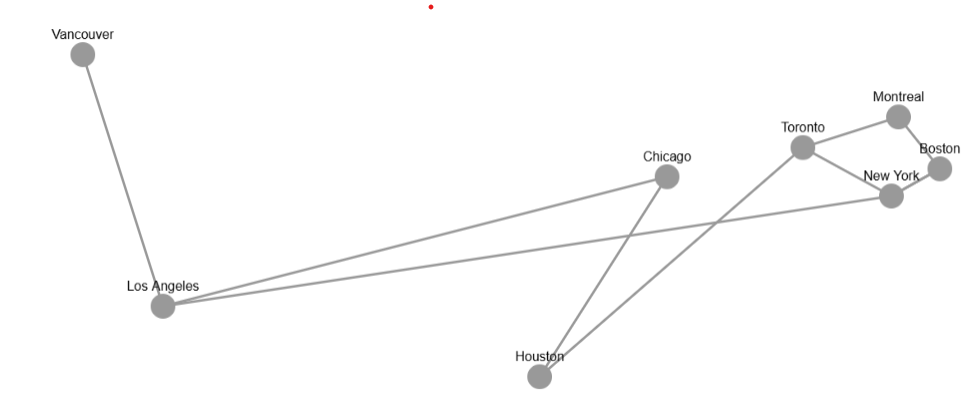


Рис.6 Пример макета Preset.

* Random:

Исходя из его названия, данный алгоритм размещения узлов осуществляет случайное распределение элементов внутри области просмотра. Несмотря на то, что он редко применяется для конечной визуализации из-за отсутствия четкой структуры, этот макет оказывается полезным для проведения тестирования и отладки.

* Grid

Этот макет предназначен для размещения узлов в стандартной сетке. Он помогает обеспечить равномерное распределение узлов и может быть особенно полезен для графов, где позиции узлов не должны отражать какие-либо внутренние отношения данных. Если при инициализации графа Cytoscape макет не задан, применяется grid, так как данный макет является простым и понятным, что делает ее предпочтительным вариантом по умолчанию.

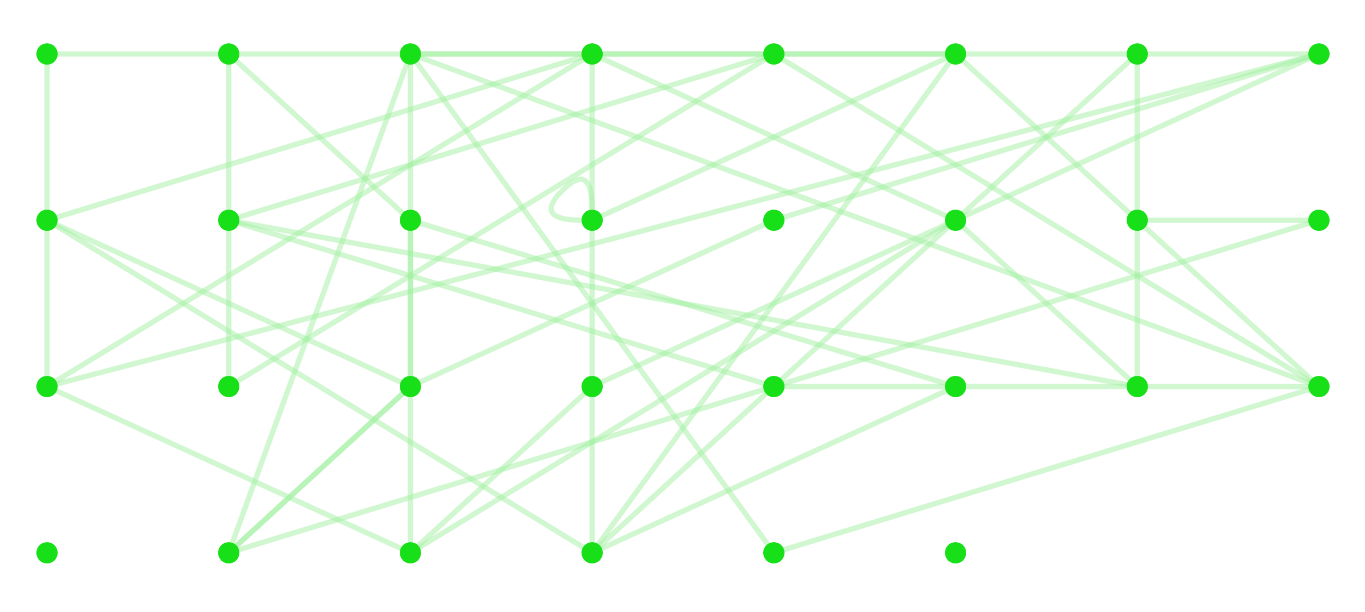


Рис.7 Пример макета Grid.

* Circle:

Макет Circle организует узлы в круг. По умолчанию узлы размещаются по часовой стрелке от 12 часов в том порядке, в котором они передаются макету. Есть возможность управлять порядком узлов, вызывая eles.sort().layout() или упорядочивая элементы по мере их добавления в граф. Это может быть особенно полезно, когда необходимо избежать пересечения рёбер между узлами или когда граф должен быть расположен вокруг центральной точки для лучшей визуализации данных или анализа.

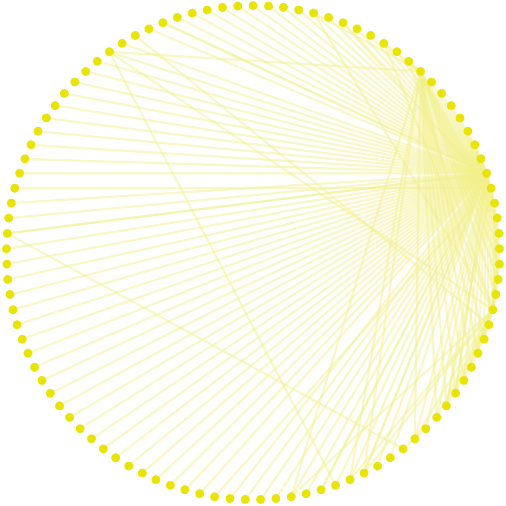


Рис.8 Пример макета circle.

* Concentric:

Concentric макет организует узлы в последовательность концентрических кругов, основываясь на определенном метрическом показателе. Узлы с наибольшими значениями метрики размещаются ближе к центру, с каждым последующим кругом от центра постепенно снижаются значение метрики узловю. Такой макет позволяет выделить относительную важность узлов, а его визуальное восприятие может быть усилено путем создания сопоставления стилей с метрикой — например, узлы с большими значениями метрики будут иметь более темный цвет.

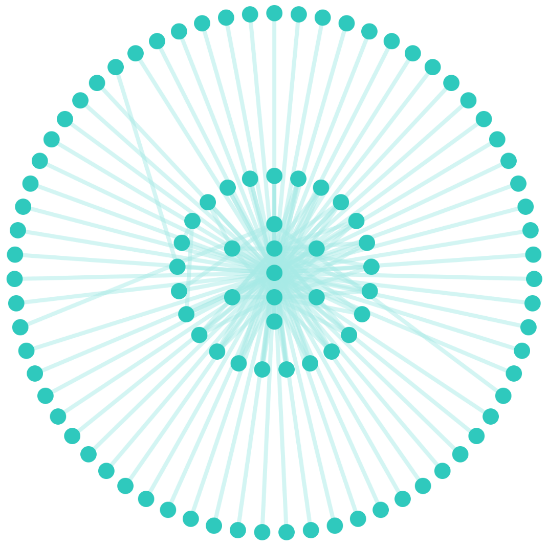


Рис.9 Пример макета concentric.

* Breadthfirst:

Схема Breadthfirst предназначена для организации узлов на уровни, основанные на их расстоянии от определенного корневого узла. Эта схема особенно хорошо подходит для визуализации деревьев и других иерархических структур, где понимание уровней иерархии имеет решающее значение.

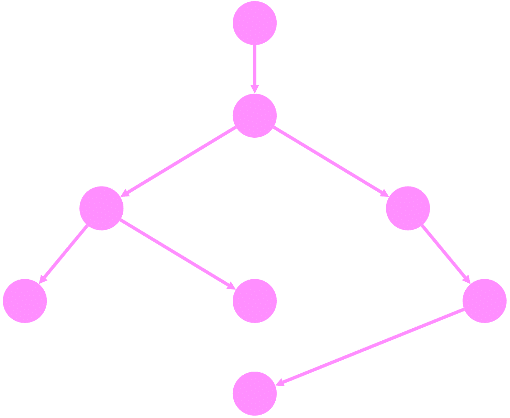


Рис.10 Пример макета Breadthfirst

* Cose: Макет Cose в Dash Cytoscape является сложным и широко используемым алгоритмом для визуализации сложных сетей. Cose означает "Compound Spring Embedder", который моделирует притяжение и отталкивание между элементами сети, основываясь на принципах, описанных в работе Dogrusoz et al., 2009. Этот макет строит графы таким образом, что узлы располагаются так, чтобы минимизировать пересечения рёбер и создать эстетически приятное расположение.

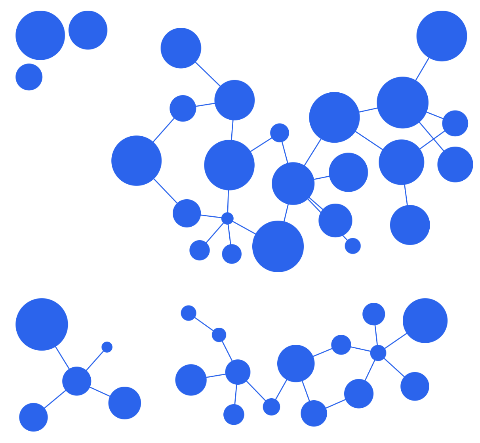


Рис.11 Пример макета Cose

Выше были описаны основные макеты Dash-cytoscape, однако с выпуском версии 0.1.1 или более поздней версией появилась возможность использовать несколько мощных внешних макетов для улучшения визуализации их графиков. С официальной библиотекой Dash Cytoscape распространяются следующие внешние макеты: cose-bilkent, cola, euler, spread, dagre, klay.

Для использования этих внешних макетов необходимо импортировать и вызывать функцию load\_extra\_layouts() из Dash Cytoscape.

Стилизация.

После того, как тема макетов была изучена, важно обратить внимание на то, что Dash-cytoscape предоставляет широкие возможности для стилизации графов. Это поможет сделать графы не только более привлекательными, но и сделать их информативными. В результате чего, легко настраиваемые под конкретные потребности исследования.

Параметра Stylesheet предназначен для определения стилей элементов графа, который принимает список словарей. Каждый словарь в этом списке должен содержать два ключа:

* 'selector': строка, указывающая, какие элементы будут стилизоваться.
* 'style': словарь, в котором указано, что именно вы хотите изменить. Это могут быть такие параметры, как ширина, высота, цвет узла, форма стрелки на ребре и многое другое.

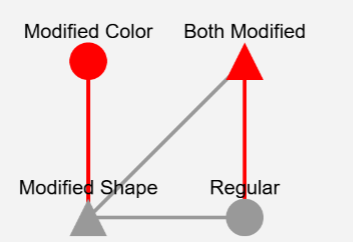
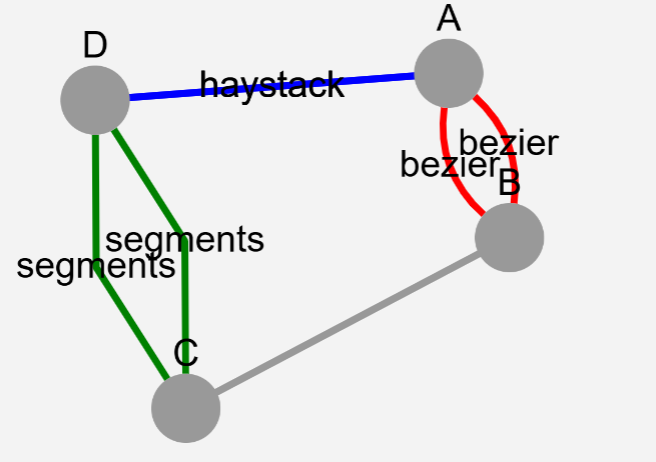
****

Рис.12 Стилизованный граф



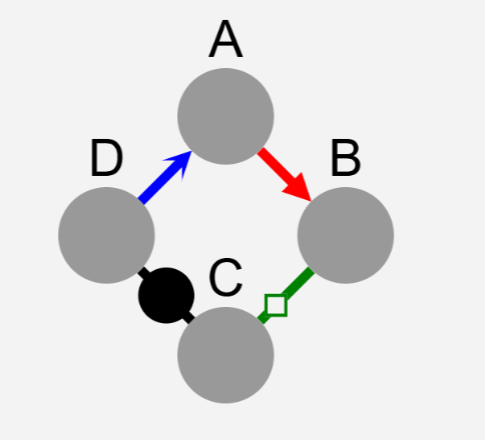
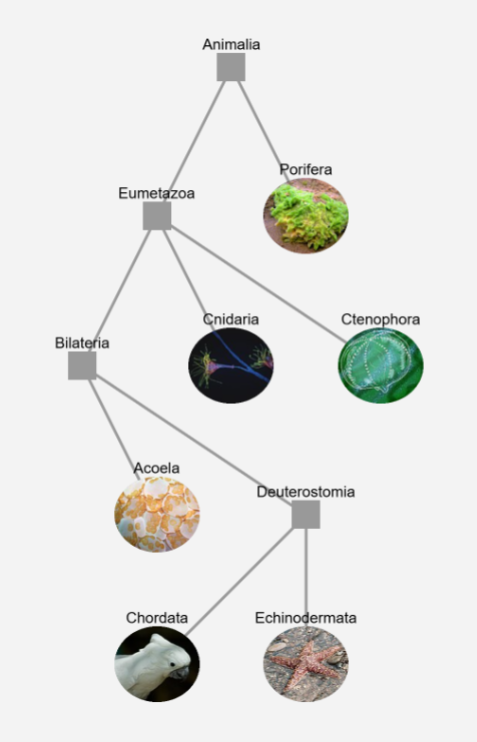


Рис.13 Пример того, как можно стилизовать ребра.



Так же есть возможность отображать изображения внутри узлов. На данном рисунке приведен полный пример отображения интерактивного дерева филогении животных с использованием изображений Викимедиа.

Callbacks.

Callbacks в Dash-Cytoscape предоставляют механизм для динамического обновления графов на основе взаимодействия пользователя с другими компонентами интерфейса, например, такими как выпадающие списки, кнопки и слайдеры, обновление стилей, добавление новых узлов и ребер, изменение макета и многое другое, что позволяет создавать интерактивные и адаптивные визуализации, которые реагировать на действия пользователя в реальном времени.

В рамках данного исследования был детально изучен инструмент Dash Cytoscape, представляющий собой инструмент для визуализации графов в веб-приложениях на платформе Dash. Благодаря своей способности интегрировать функциональность библиотеки Cytoscape.js, Dash Cytoscape позволяет создавать высокопроизводительные, интерактивные и настраиваемые графовые визуализации, что делает его незаменимым для анализа сетевых структур данных.

# **2.РАЗРАБОТКА МОДУЛЯ.**

Цель разработки модуля заключается в создании инструмента для ввода ответа обучаемого в виде графа, который может быть использован на образовательных платформах. Модуль должен обеспечивать:

* Возможность создания и редактирования графов.
* Сохранение графов в формате JSON.
* Возможность загружать ответ обучаемого.

## Архитектура модуля.

Архитектура модуля включает следующие основные компоненты:

* Клиентская часть: Интерфейс пользователя, реализованный с помощью Dash, включает в себя элементы управления для создания и редактирования графов.
* Серверная часть: Обработка запросов от клиента, валидация данных и их сохранение.
* Возможность сохранять: Хранение информации о графах и их элементах.
* Визуализировать сохраненный файл, для последующего редактирования.

**Язык программирования и IDE.**

Для разработки модуля были выбраны язык программирования Python и интегрированная среда разработки PyCharm. Этот выбор обусловлен следующими причинами:

Язык программирования Python широко используется благодаря своей простоте и читаемости, что ускоряет процесс разработки и упрощает поддержку кода. Python имеет обширный набор библиотек и фреймворков, например, таких как Dash. Так как, Dash фреймворк является достаточно мощным инструменты для построения, анализа и визуализации графов, делает Python идеальным выбором для данной задачи.

Интегрированная среда разработки (IDE) PyCharm предоставляет множество функций, облегчающих разработку на Python. PyCharm обладает интуитивно понятным интерфейсом и широким спектром инструментов, что делает его удобным для использования как начинающими, так и опытными программистами. Основные возможности PyCharm включают поддержку отладки, интеллектуальную подсветку синтаксиса, автодополнение кода, а также интеграцию с системами контроля версий, такими как Git. Мой выбор данной среды обусловлен опытом работы с PyCharm, а также его функциональными возможностями, которые значительно ускоряют процесс разработки и повышают его эффективность.

**Реализация модуля.**

Первым шагом к созданию модуля является импорт необходимых библиотек и инициализация.

Из ниже представленного фрагмента кода становится ясно, что разработка осуществлялась с применением библиотеки Dash, которая позволяет разрабатывать веб-приложения на языке Python. В состав основных компонентов приложения входят Dash, Dash Cytoscape и Dash Bootstrap Components. Кроме того, был задан стиль веб-приложению и имя, которое будет отображаться на вкладке Браузера.

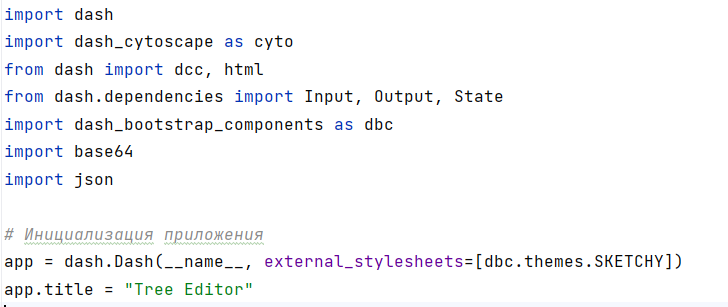


Рис. 14 Фрагмент кода

Возможность выбора цветов для узлов и ребер.

В этой главе будет упомянуто функциональность выбора цветов для узлов и ребер. Вначале представляется предварительный список цветов, который впоследствии реализуется через выпадающий список с различными вариантами цвета. Приведенный ниже фрагмент кода показывает текущий набор доступных цветов, однако существует возможность значительно расширить этот список.



Рис.15 Список доступных цветов.

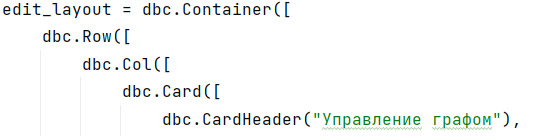
Далее разрабатывается интерфейс пользователя, который состоит из двух основных компонентов, — это панели управления и области отображения графа.  
  


Рис.16 Контейнер «Управление графом»

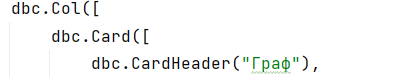


Рис.17 Контейнер «Граф»

Панель управления включает следующие элементы и возможности:

* Выбор направления растущего дерева. Пользователь может выбрать направление роста дерева между двумя вариантами: сверху вниз (TB) или снизу вверх (BT). Это влияет на то, какой граф будет строиться, семантический или аналитический. Данная функция перестает быть активным после добавления корневого узла.
* Добавление и стилизация корневого узла. Пользователь может вводить название корневого узла через текстовое поле, выбирать цвет для корневого узла из выпадающего списка с различными цветами. Так же при желании может загрузить изображение для корневого узла через элемент dcc.Upload, который становится видимым после выбора опции «Работа с изображениями» в dcc.Checklist.
* Добавление потомка. После добавления корневого узла активируется кнопка "Добавить потомка", которая позволяет добавить дочерний узел к корню. Пользователь может вводить название и метку для потомка, выбирать цвет для узла и ребра между родителем и потомком. Как и корневой узел, дочерние узлы поддерживают загрузку изображения для узла через элемент dcc.Upload, который становится видимым при добавлении узла, после активации «Работы с изображением».
* Удаление узла. После выбора узла нажатие на кнопку "Удалить узел" удалит выбранный узел из графа, это может быть как корневой, так и дочерний узел, если он был добавлен в граф.
* Очистка графа. Пользователь может полностью очистить граф, удалив все узлы и связи, нажав на кнопку "Очистить граф" и начать построение графа заново.
* Сохранение графа. Пользователь может сохранить текущее состояние графа в файл JSON, введя имя файла в соответствующее текстовое поле и нажав на кнопку "Сохранить граф".
* Загрузка дерева. Пользователь может загрузить дерево из файла JSON, используя элемент dcc.Upload. После загрузки дерева оно автоматически отображается в области отображения графа.
* Видимость элементов управления. Некоторые элементы управления, такие как поля для загрузки изображений, становятся видимыми или невидимыми в зависимости от действий пользователя, например, выбора опции "Работа с изображениями".

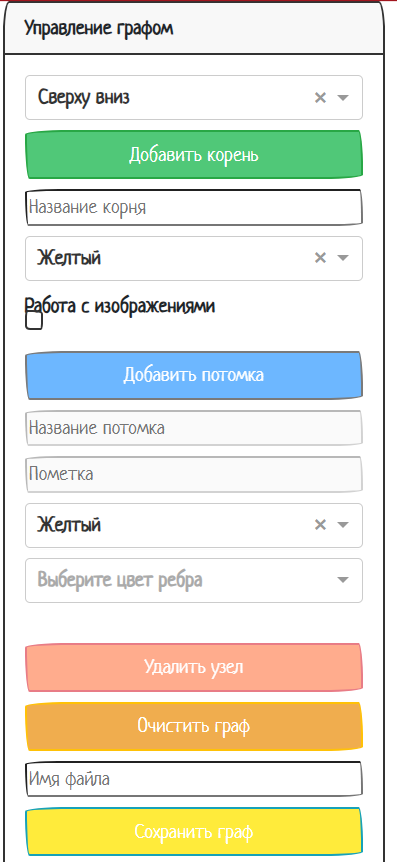
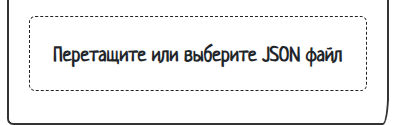


Рис.18 Панель управления графом.

Отображение графа включает в себя следующие элементы и возможности:

* Визуализация графа. Используется библиотека Dash-Cytoscape для отображения графа внутри заданного HTML-контейнера. Граф строится на основе списка элементов (elements), включающего узлы и ребра, каждый из которых имеет свои данные и стили.
* Стилизация графа. Для каждого типа элементов графа (узлы и ребра) задаются стили, определяющие их внешний вид. Например, можно установить фон, форму, размер, цвет и другие параметры узлов и ребер. Эти стили применяются к элементам графа через селекторы в таблице стилей (style).
* Макеты (Layout) расположения. Dash-Cytoscape позволяет использовать различные алгоритмы макетов для автоматического расположения узлов на плоскости. Это помогает организовать граф таким образом, чтобы он был понятным и эстетичным. В примере используется макет dagre, который подходит для отображения деревьев и других направленных графов.
* Интерактивность. Пользователи могут взаимодействовать с графом, например, перемещать узлы, увеличивать и уменьшать масштаб, а также выбирать узлы и ребра для дальнейшего редактирования. Dash-Cytoscape поддерживает множество жестов и действий мыши для обеспечения удобства использования.
* Редактирование свойств элементов. В области отображения графа пользователи могут изменять свойства выбранных узлов и ребер, такие как метки и цвета. Это позволяет динамически модифицировать граф без необходимости перезагрузки страницы или удаления и полного создания заново.
* Отображение дополнительной информации: В зависимости от выбранных узлов и ребер, в интерфейсе могут отображаться дополнительные поля ввода для изменения меток и других атрибутов. Это обеспечивает гибкость в работе с данными графа, например, если пользователь допустил ошибку в названии меток, нет необходимости заново рисовать граф.



Рис.19 Область отображения графа.

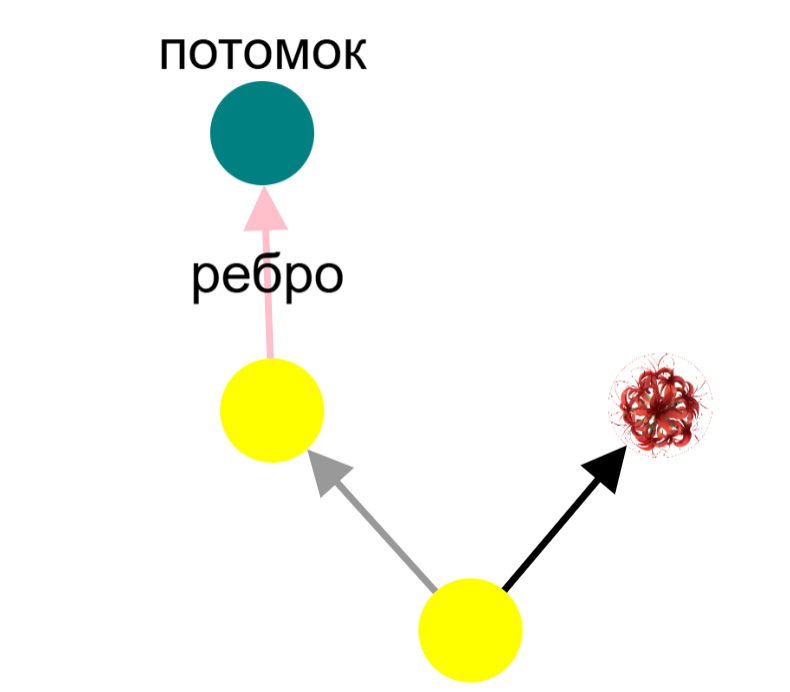


Рис.20 Простая демонстрация работающих функций модуля.

Из данного рисунка видно, что работают функции выбора направления графа, выбора цветов для узлов и ребер, возможность добавить фон узлам с помощью загрузки изображения.

Обратные вызовы (callbacks).

Обработка событий и обновление графа в данном модуле осуществляется через обратные вызовы (callbacks), реагируя на изменения состояния интерфейса пользователя, таких как нажатия кнопок, выборы из выпадающих списков и загрузка файлов, и обновляют граф соответственно.

Рассмотрим несколько ключевых моментов обработки событий и обновления графа:

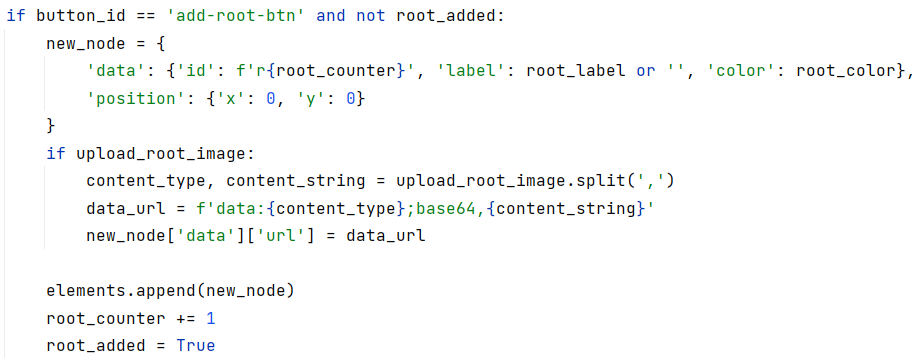
Callback для добавления корневого узла. Когда пользователь нажимает кнопку "Добавить корень", вызывается callback, который проверяет, не был ли уже добавлен корневой узел (root\_added). Если нет, то создается новый узел с заданным названием и цветом, а также позицией (в данном случае, начальная позиция находится в начале координат). Если пользователь загрузил изображение для корневого узла, оно также добавляется к узлу.  


Рис. 21 Фрагмент кода логики добавления корневого узла.

Callback для добавления потомка. После добавления корневого узла активируется кнопка "Добавить потомка". При нажатии на эту кнопку вызывается callback, который добавляет новый узел как потомка к последнему выбранному узлу. Затем создается ребро, связывающее родительский узел с новым узлом.



Рис. 22 Фрагмент кода логики добавления дочернего узла.

Callback для удаления узла. Callback, отвечающий за удаление узла, активируется при нажатии на кнопку "Удалить узел", в следствии чего удаляется выбранный узел из графа, а также любые ребра, связанные с этим узлом.

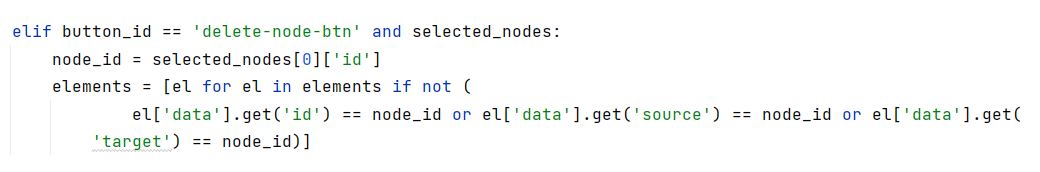


Рис. 23 Фрагмент кода логики удаления узлов.

Callback для очистки графа. При нажатии на кнопку "Очистить граф" вызывается callback, который очищает все узлы и ребра из графа, устанавливая список элементов обратно в пустой массив.

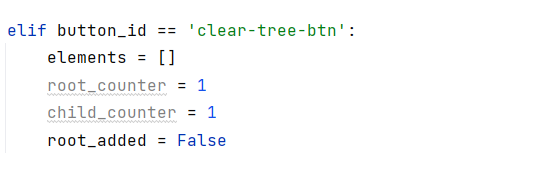


Рис. 24 Фрагмент кода логики удаления графа.

Callback для сохранения и загрузки дерева. Callbacks для сохранения и загрузки дерева позволяют пользователю сохранять текущее состояние графа в файл JSON и загружать дерево из файла JSON соответственно. Это позволяет сохранять прогресс работы над деревом и восстанавливать его позже, для дальнейшего взаимодействия с ним.

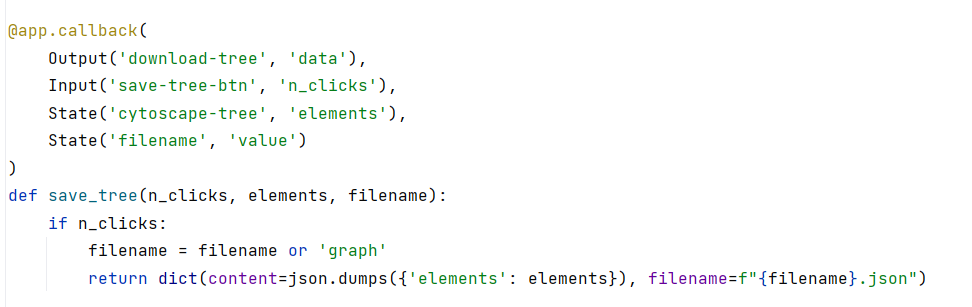


Рис.25 Фрагмент кода логики сохранения графа.

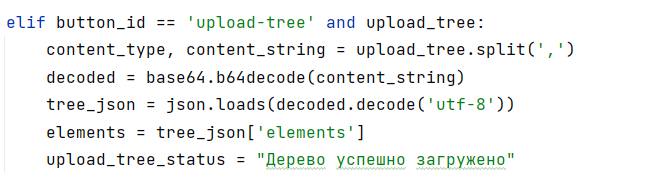


Рис.26 Фрагмент кода логики загрузки из файла.

Callback для обновления стилей и поведения графа. Callback, связанный с выбором направления растущего дерева, позволяет динамически изменять стиль и поведение графа в зависимости от выбранного направления роста.

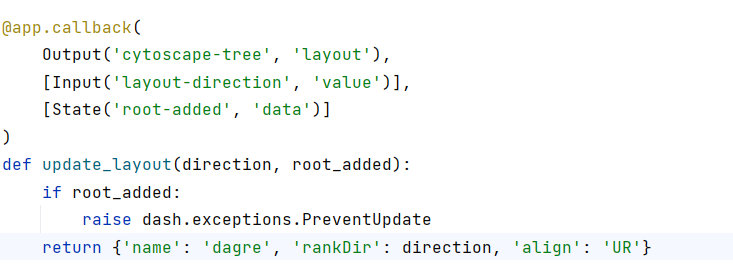
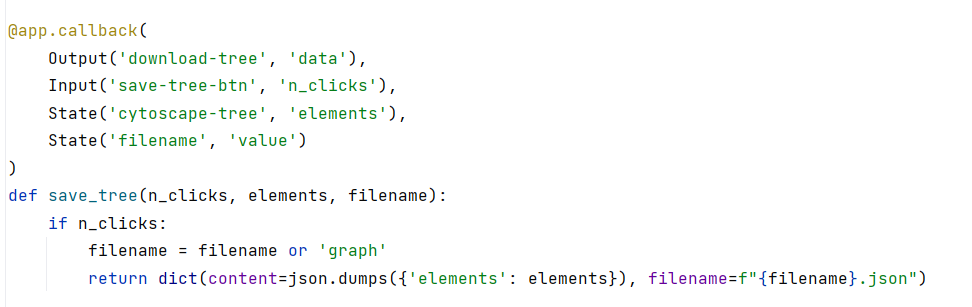


Рис.27 Фрагмент кода с логикой задания макета.

Все эти callbacks обеспечивают взаимодействие между пользовательским интерфейсом и графическим представлением дерева, позволяя пользователю эффективно создавать, редактировать и управлять графовыми деревьями.

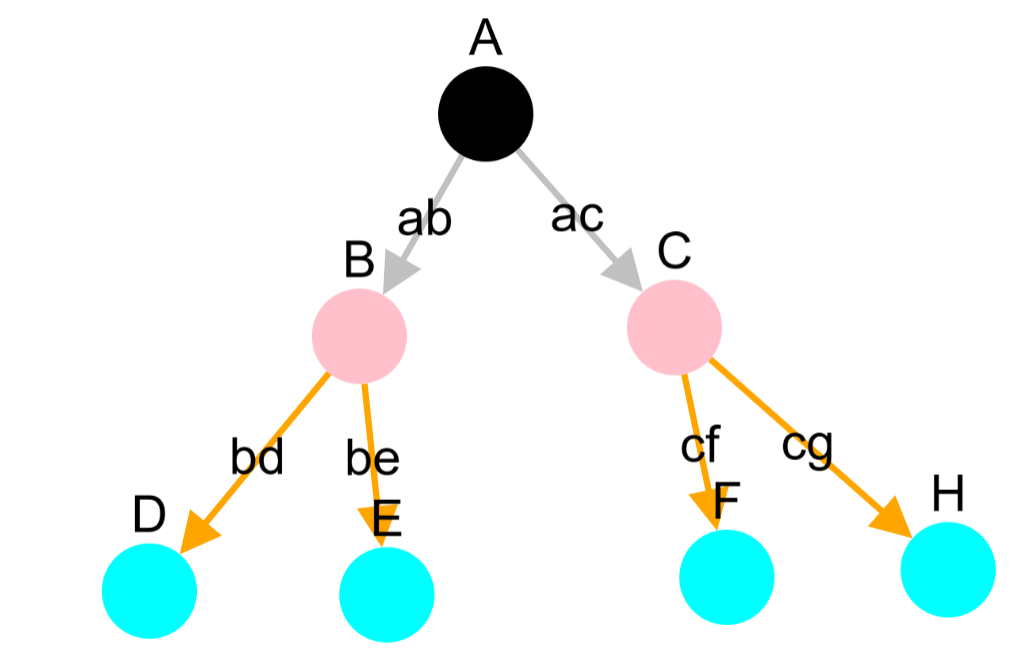
Сохранение и загрузка графов.

Процесс сохранения дерева начинается с кнопки "Сохранить граф". Когда пользователь нажимает эту кнопку, вызывается callback-функция save\_tree. Эта функция принимает три параметра: количество кликов по кнопке "Сохранить граф" (n\_clicks), текущие элементы дерева (elements) и имя файла (filename). Если пользователь нажал кнопку хотя бы один раз, функция сохраняет текущее состояние дерева в формате JSON и возвращает объект, содержащий этот JSON и имя файла. Этот объект затем передается в компонент dcc.Download, который генерирует ссылку для скачивания файла.



Важно отметить необходимость ввода имени файла в поле задания названия файлу JSON filename, перед нажатием на кнопку "Сохранить граф". Однако если имя файла не указано, будет использоваться значение по умолчанию 'graph.json'.

Например, создадим произвольное грфовое дерево и сохраним его, придав название «Произвольный граф».

  
Рис.28 Произвольное дерево

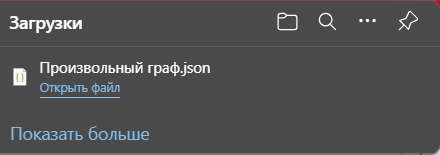


Рис. 29 Загрузка дерева

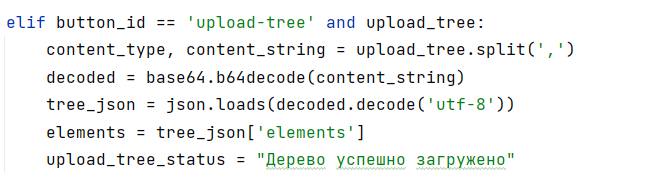
Таким образом скачивается файл, если открыть, его содержимое выглядит таким образом:



Каждая вершина и ребро представляются как отдельный объект в массиве "elements". Каждый объект содержит набор свойств, описывающих характеристики конкретного элемента. Например, свойства "id", "label" и "color" используются для идентификации вершины, её метки и цвета соответственно, а свойства "source" и "target" указывают на начало и конец ребра. Позиция каждого элемента на плоскости определяется координатами "x" и "y".

Сохранения графовых структур в формате JSON играет критическую роль в современных веб-приложениях и системах, обеспечивая универсальный и гибкий способ обмена данными между различными платформами и языками программирования. Использование JSON позволяет не только упрощать сложные документы до значимых компонентов, но и облегчает манипуляцию и извлечение данных, поддерживая вложенные структуры и массивы, что может быть весьма полезным при анализе графов.

Процесс загрузки графового дерева реализован через callback-функцию update\_graph, которая обрабатывает несколько различных действий пользователя, включая загрузку дерева из JSON файла.



Этот фрагмент кода выполняет следующие действия проверяет, был ли выполнен запрос на загрузку дерева, разделяет содержимое загруженного файла на тип контента и строку содержимого, декодирует содержимое файла из Base64, преобразует декодированную строку в объект Python с помощью json.loads, извлекает список элементов дерева из объекта и присваивает его переменной elements, и наконец обновляет статус загрузки дерева сообщением "Дерево успешно загружено".

Такой процесс позволяет пользователям загружать существующие деревья в приложение, сохраняя возможность дальнейшего редактирования и визуализации этих деревьев с помощью Dash-Cytoscape.

**Проблема проверки правильности решения заданий с ответом, представленным в виде графа.**

Сначала предполагалось сравнивать построенный граф с некоторым эталоном, например определять, изоморфны ли эталонный и построенный графы. В дальнейшем от решения этой задачи по предложению научного руководителя решено было отказаться.

Главная причина отказа от поддержки такой функции – это тот факт, что правильных решений задачи может быть несколько (а иногда и бесконечно много), и графы-решения при этом могут и не быть изоморфными.

Поэтому анализ решения задачи, должен зависеть от характера этой задачи, а приложение, проверяющее правильность решения должно разрабатываться отдельно и быть, например, консольным приложением, проверяющим решения студентов некоторой группы, собранных в одной рабочей папке.

В качестве примера, предлагается консольное приложение, проверяющее построение семантических деревьев.

## ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ.

**Примеры задач на который ученик будет писать ответ в виде графа.**

Задача №1.

С помощию резолюции доказать, что следующее множество дизъюнктов невыполнимо:

P ∨ Q ∨ R, ~P ∨ R, ~Q, ~R.

|  |  |  |
| --- | --- | --- |
| 1 | P V Q V R | Hypothesis |
| 2 | ~P V R | Hypothesis |
| 3 | ~Q | Hypothesis |
| 4 | ~R | Hypothesis |
| 5 | Q V R | 1, 2 |
| 6 | R | 3, 5 |
| 7 | ☐ | 4, 6 |

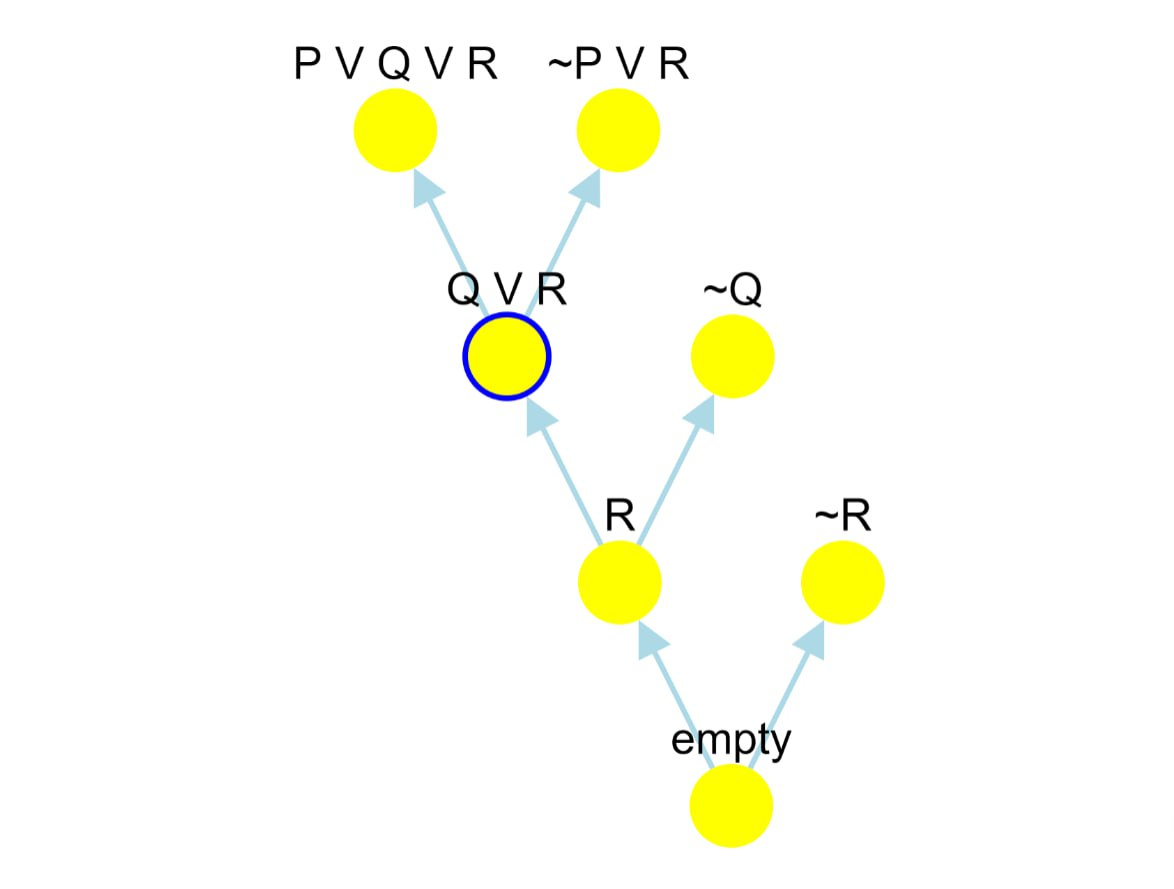


Рис. 30 Дерево вывода. Ответ обучаемого, представленный в виде графа.

Задача №2.

С помощью резолюции для множества S = {P ∨ Q, ~ Q ∨ R, ~P ∨ Q, ~R} получить пустой дизъюнкт из S.

|  |  |  |
| --- | --- | --- |
| 1 | P V Q | Hypothesis |
| 2 | ~Q V R | Hypothesis |
| 3 | ~P V Q | Hypothesis |
| 4 | ~R | Hypothesis |
| 5 | P V R | 1, 2 |
| 6 | Q V R | 3, 5 |
| 7 | R | 2, 6 |
| 8 | empty | 4, 7 |

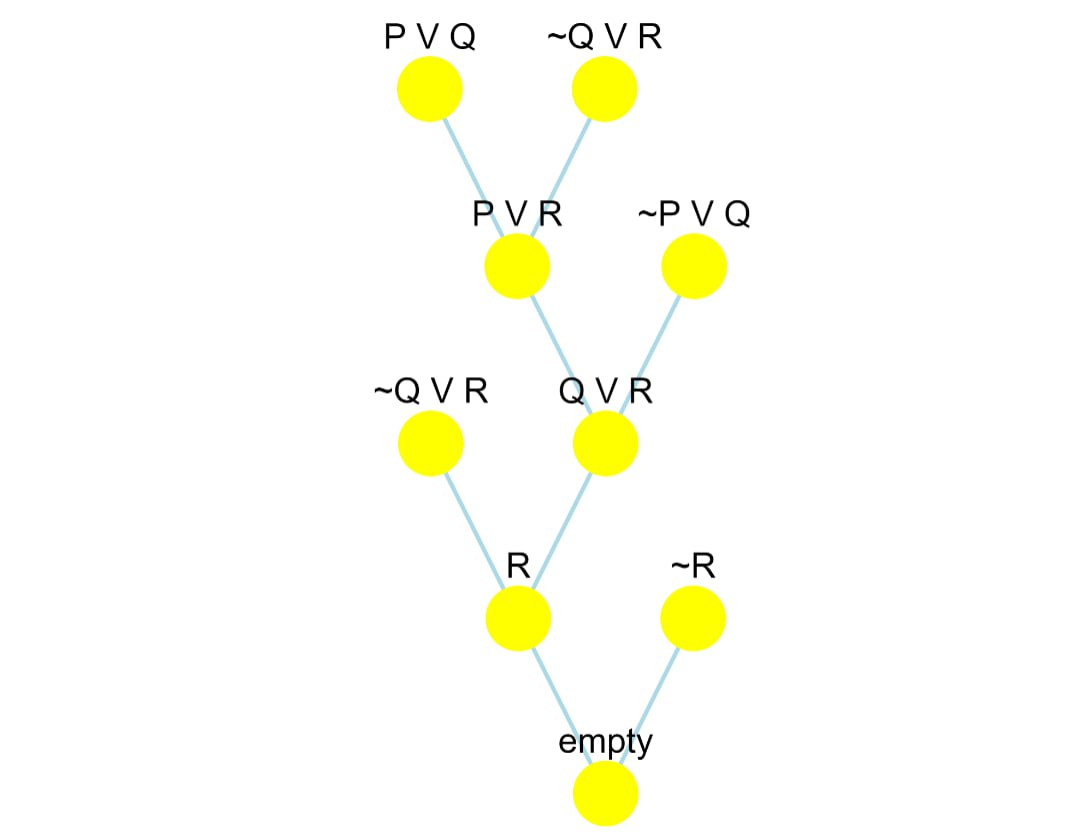


Рис. 31 Ответ обучаемого, представленный в виде графа.

Задача №3.

Пусть S = {P, ~ P ∨ Q, ~Q}. Построить замкнутое семантическое дерево для S.

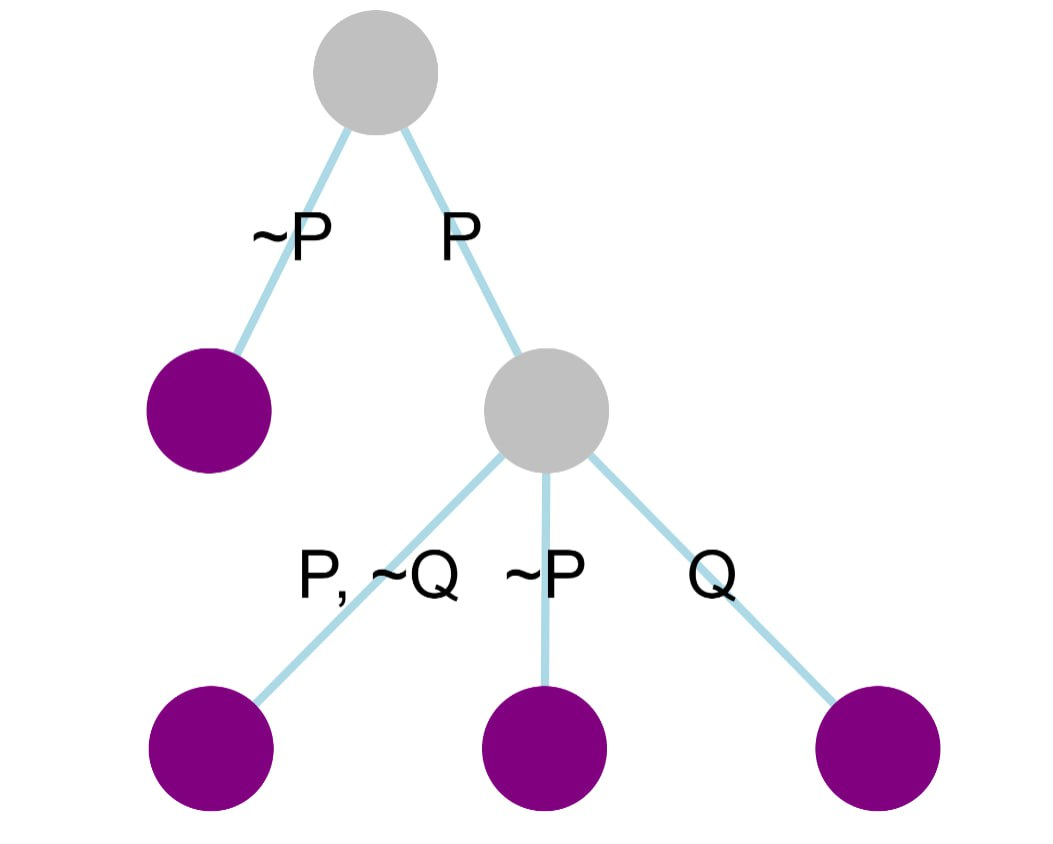


Рис.31 Ответ обучаемого.

\* Фиолетовым обозначены опровергающие узлы

**Примеры графов для иллюстрации.**

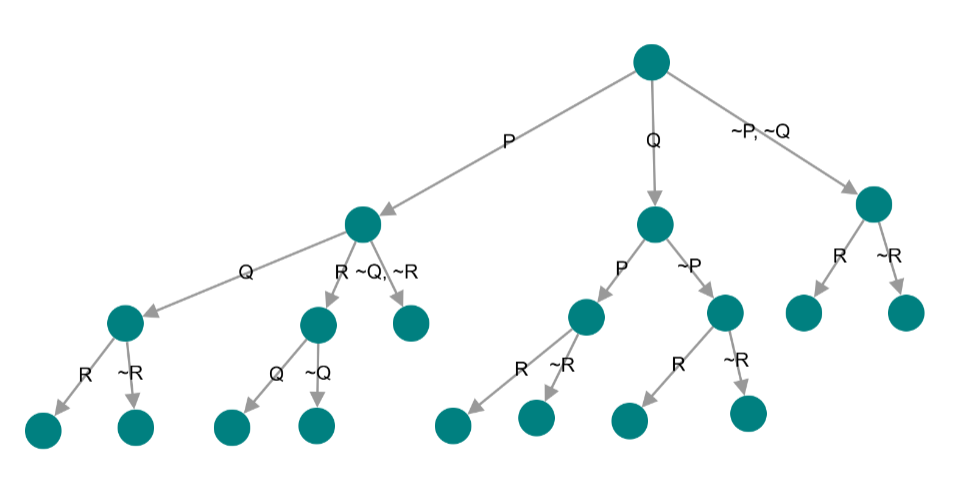
****

Рис.33 Полное семантическое дерево.

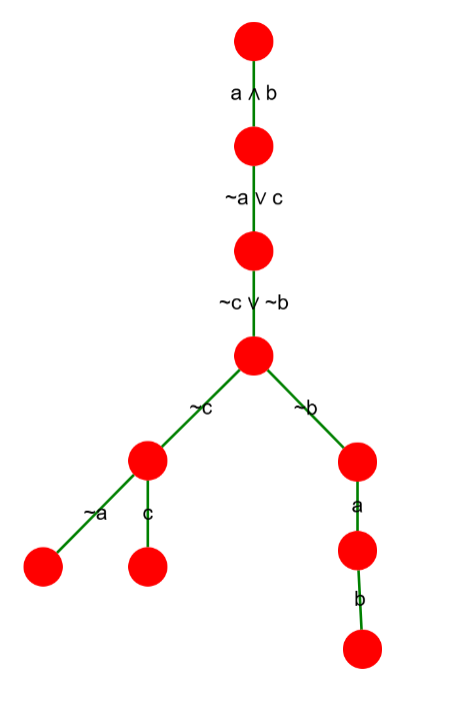


Рис.34 Графическое построение частично построенной таблицы предложений.

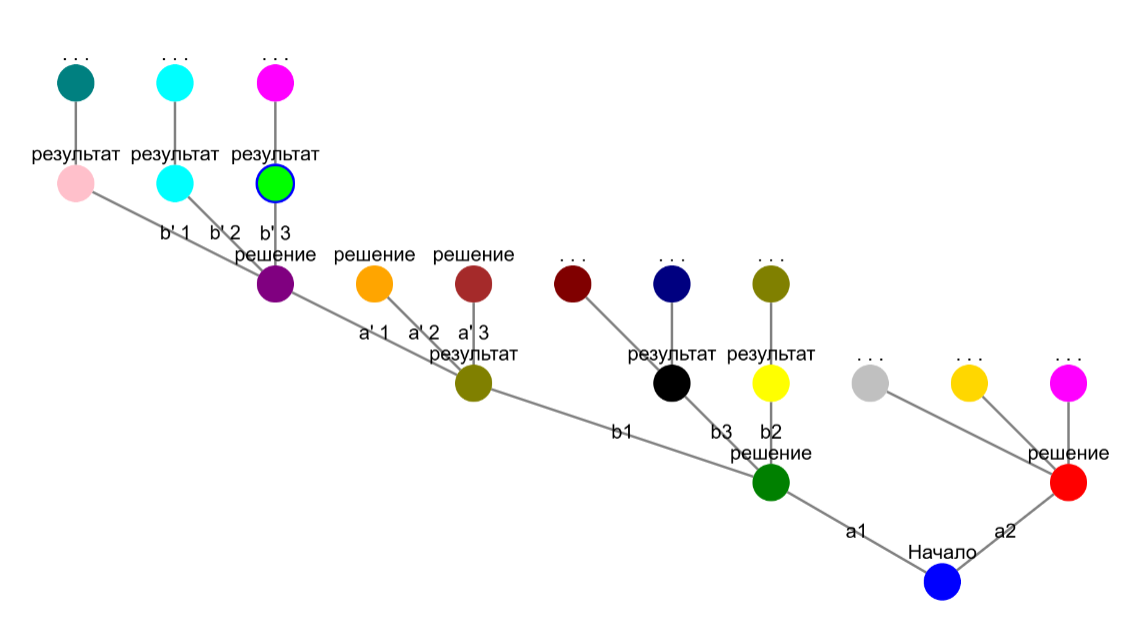


Рис.35 Аналитическое дерево решений.

а – варианты действий

b – возможные последствия выбранных действий

# **ЗАКЛЮЧЕНИЕ.**

В ходе выполнения данной выпускной квалификационной работы был разработан модуль для визуализации и редактирования графов с использованием библиотеки Dash и ее компонента Dash-Cytoscape. Эта система позволяет создавать, редактировать, сохранять и загружать графовые структуры, что значительно упрощает процесс взаимодействия пользователей с данными, представляемыми в виде графов.

Разработка модуля ввода ответов обучаемых в виде графов отвечает современным требованиям образовательного процесса. Использование графов для визуализации и структурирования данных способствует лучшему усвоению информации и повышению интерактивности учебных материалов.

Достигнутые задачи:

* Разработан интерфейс, который позволяет пользователям легко создавать и редактировать графы. Это включает в себя добавление узлов и рёбер, задание их свойств и стилей, а также загрузку изображений для узлов.
* Реализована возможность сохранения созданных графов в формате JSON, что позволяет пользователям продолжать работу с графами в любое время без потери данных. Также обеспечена функция загрузки графов из JSON-файлов, что делает модуль гибким и удобным в использовании.
* Включены различные элементы управления, такие как выпадающие списки, кнопки и поля ввода, которые упрощают процесс создания и редактирования графов. Это позволяет пользователям интуитивно взаимодействовать с интерфейсом.
* Модуль позволяет визуально представить сложные взаимосвязи между элементами данных, что улучшает процесс обучения и помогает студентам лучше усваивать материал.
* Универсальность применения. Разработанный модуль может быть использован в различных дисциплинах, таких как математика, информатика, биология и социология, что делает его универсальным инструментом для образовательных целей.
* Интерактивность и вовлечённость. Студенты могут активно взаимодействовать с графами, что способствует лучшему усвоению материала и повышает их вовлечённость в учебный процесс.

Перспективы дальнейшего развития:

* Расширение функционала. Введение дополнительных инструментов для анализа графов, таких как алгоритмы поиска кратчайшего пути, определения центральности узлов и т. д. Это позволит использовать модуль для решения более сложных задач и углубленного изучения теории графов.
* Интеграция с другими образовательными платформами. Возможность интеграции разработанного модуля с популярными образовательными платформами и системами управления обучением.
* Улучшение визуализации. Введение новых методов и стилей визуализации для более наглядного и привлекательного представления графов, включая анимации и интерактивные диаграммы.

# **ЛИТЕРАТУРА** **& ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ**.

1. [Dash Documentation & User Guide | Plotly](https://dash.plotly.com/)
2. The Book of Dash. 2022 by Adam Shroeder, Christian Mayer, and Ann Marie Ward.
3. [Using layouts · Cytoscape.js](https://blog.js.cytoscape.org/2020/05/11/layouts/#layout-definition)
4. Алексеев В. Е., Захарова Д. В. Теория графов: учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017.
5. Харари Ф. Теория графов. — М.: УРСС, 2003.
6. Математическая логика и автоматическое доказательство теорем. Ч. Чень, Р.Ли. Под редакцией С. Ю. Маслова 1983.
7. [Method of analytic tableaux - Wikipedia](https://en.wikipedia.org/wiki/Method_of_analytic_tableaux#:~:text=An%20analytic%20tableau%20is%20a,or%20refute%20the%20whole%20formula)
8. Formal Logic. Its scope and Limits. Fourth Edition by Richard Jeffrey. Год публикации 1984
9. Handbook of automated reasoning. Alan Robinson and Andrei Voronkov. 2001 год.
10. Карпов, Д. В. Теория графов. Москва: Издательство Наука, 2005.