МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»**

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК (ИКН)

Курсовая работа на тему

«Поиск мостов в графе»

Выполнил: студент группы БИВТ-23-6

Пешков Матвей Евгеньевич

Москва, 2025

**Введение**

Графы — это важный объект в информатике и смежных науках, позволяющий моделировать множество реальных систем: от компьютерных сетей и социальных связей до электрических схем и дорог. Одним из ключевых элементов в теории графов являются **мосты** — такие рёбра, удаление которых увеличивает количество компонент связности графа. Другими словами, мост — это критическое соединение, без которого нарушается целостность структуры.

Задача **поиска мостов** в неориентированном графе имеет важное прикладное значение. Например:

* В компьютерных сетях мосты могут указывать на критические соединения, отказ которых приведёт к разрыву связи.
* В проектировании дорожных сетей важно знать, какие дороги являются единственным соединением между районами.
* В социальных графах мост может указывать на уникальную связь между сообществами.

В данной работе реализован алгоритм поиска мостов в графе, основанный на углублённом поиске в глубину (**DFS**). Алгоритм обладает линейной сложностью по числу вершин и рёбер и эффективен даже для больших графов. Кроме того, в проекте реализована визуализация, автоматическое тестирование и структура, позволяющая легко использовать и модифицировать код.

**Теоретическая часть**

Мост в графе — это ребро, удаление которого приводит к увеличению числа компонент связности. Иначе говоря, если удалить мост, то граф распадается на части. Формально, для неориентированного графа G=(V,E), ребро (u,v)∈E является мостом, если после его удаления количество компонент связности увеличивается.

Для поиска мостов в графе используется модифицированный алгоритм поиска в глубину (**DFS**, depth-first search). В процессе обхода графа поддерживаются два массива:

* tin[v] — время входа вершины v в стек обхода;
* low[v] — минимальное tin достижимое из вершины v или её потомков по дереву DFS, используя не более одного обратного ребра.

Если в процессе обхода графа обнаруживается, что для некоторой вершины u, которая является потомком вершины v, выполняется условие:

low[u] > tin[v]

то ребро (v,u) является мостом. Это означает, что из поддерева вершины uuu нельзя попасть обратно в более ранние вершины графа, минуя v.

Алгоритм имеет линейную временную сложность **O(V + E)**, что делает его применимым к большим графам.

**Структура проекта и реализация алгоритма**

Проект реализован на языке Python и состоит из нескольких модулей, каждый из которых выполняет отдельную функцию. Это позволяет удобно разделять логику, тестировать и использовать код повторно.

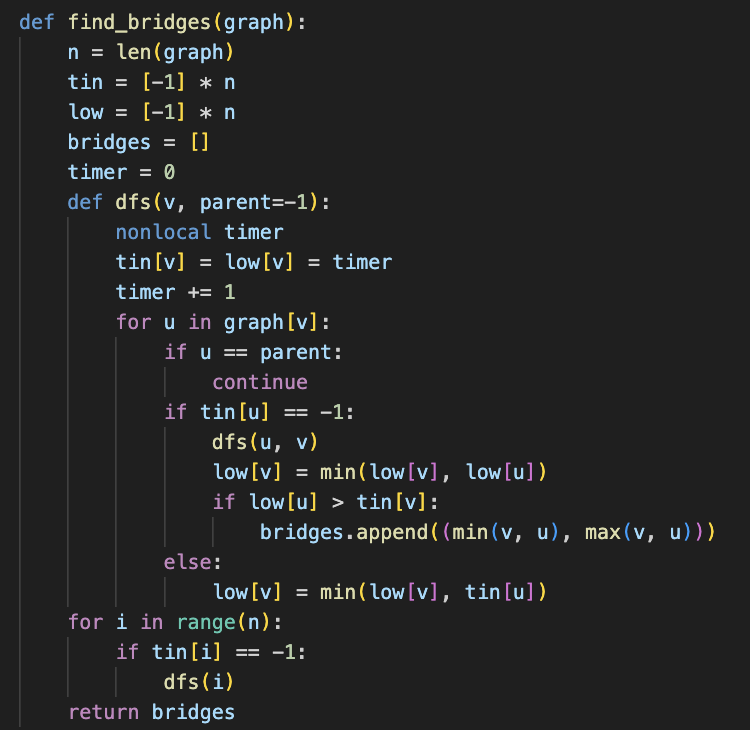
Структура проекта:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

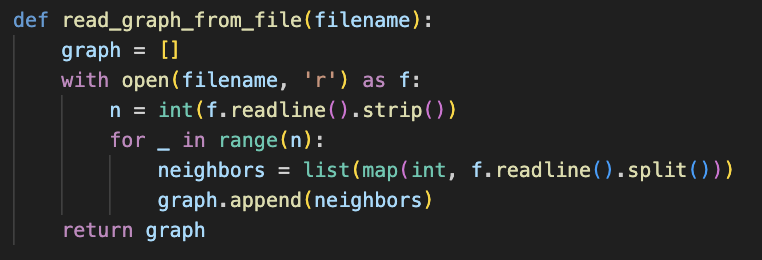
Контент, сгенерированный ИИ, может содержать ошибки.

**Реализация алгоритма (bridge.py)**

Алгоритм реализован в виде функции find\_bridges(graph), которая принимает список смежности и возвращает список мостов. Он основан на DFS и сохраняет для каждой вершины время входа и наименьшее достижимое время (tin и low). Рёбра, удовлетворяющие условию low[u] > tin[v], считаются мостами.



Дополнительно реализована функция read\_graph\_from\_file, позволяющая читать граф из текстового файла. Граф задаётся в виде количества вершин и списка смежности для каждой из них.



**Визуализация и пример использования**

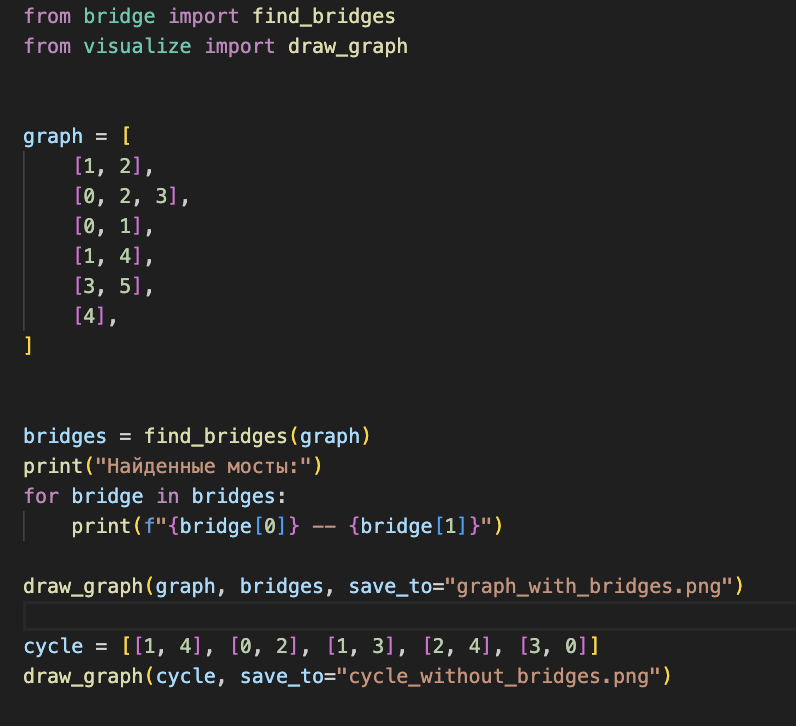
Для удобства анализа результатов реализована визуализация графа с выделением мостов. Визуализация построена с использованием библиотек networkx и matplotlib. Модуль visualize.py содержит функцию draw\_graph, которая принимает на вход граф в виде списка смежности, а также (опционально) список мостов. Граф отображается в виде вершин и рёбер, причём мосты выделяются красным цветом.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Пример использования (example.py)**

В модуле example.py демонстрируется работа алгоритма на простом графе. В коде представлен список смежности и вызов функций поиска мостов и их визуализации:

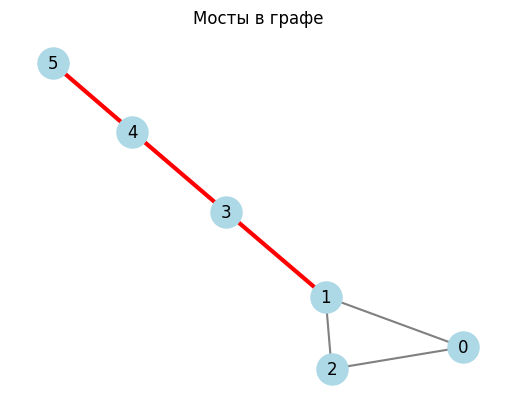


Граф до построения мостов:

Изображение выглядит как линия, круг, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

В результате выполнения будет создано изображение графа, в котором мосты выделены красным.



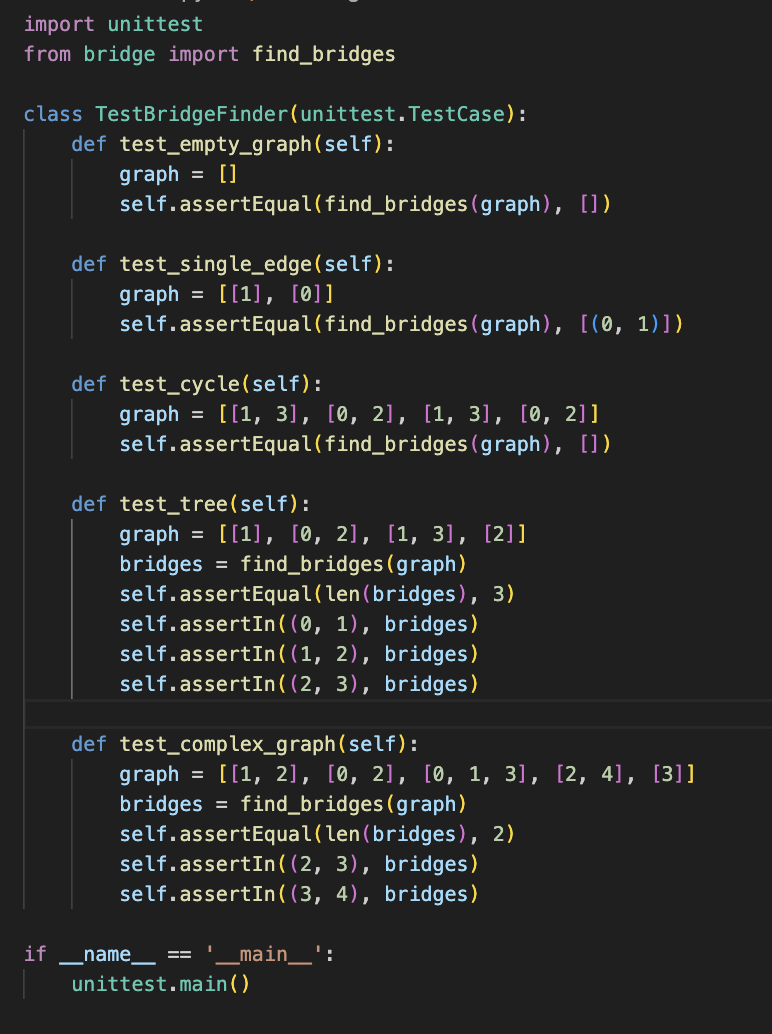
**Автоматическое тестирование и проверка корректности**

Для надёжной и корректной работы любого алгоритма важно убедиться, что он обрабатывает как стандартные, так и граничные случаи. В данном проекте реализован набор **автоматических тестов**, позволяющих убедиться в правильности работы алгоритма поиска мостов. Это особенно важно, поскольку ошибки в реализации могут привести к неверному анализу структуры графа, что критично в практических задачах — от проектирования сетей до анализа социальных структур.

Модуль test.py использует стандартную библиотеку unittest и содержит несколько тестов для типичных ситуаций:

* **Пустой граф** — не содержит вершин и рёбер, мостов быть не может.
* **Граф из одного ребра** — это ребро является мостом, так как его удаление разрывает граф.
* **Циклический граф** — например, квадрат, не содержит мостов, так как удаление любого ребра не нарушает связность.
* **Дерево** — в любом дереве каждое ребро является мостом, потому что нет альтернативных путей.
* **Сложный граф** — комбинация циклов и цепочек, включающая мосты и не-мосты.

Пример тестов:



Тесты позволяют убедиться в корректности реализации алгоритма, и могут быть запущены в любой момент, например при модификации логики или добавлении новых функций. Это особенно важно при работе в команде или использовании кода как части более крупного проекта. Автоматизация проверки даёт уверенность в том, что базовый функционал не нарушается при дальнейшем развитии проекта.

**Зависимости, запуск и документация проекта**

Для корректной работы проекта необходимо установить несколько внешних библиотек, которые используются для визуализации и построения графов. Все зависимости указаны в файле requirements.txt, который позволяет автоматически установить всё необходимое одной командой. Это удобно и ускоряет развёртывание проекта в новой среде.

Содержимое файла requirements.txt:

Изображение выглядит как текст, Шрифт, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Запуск кода**

После установки зависимостей можно запустить демонстрационный пример, который находится в файле example.py. Он создаёт небольшой граф, выполняет поиск мостов и сохраняет визуализацию:

python example.py

После запуска в рабочей директории появится изображение graph\_with\_bridges.png, на котором отображены вершины, рёбра и найденные мосты (красным цветом).

**Структура проекта**

Для удобства работы и навигации по коду проект организован по модулям:

* **bridge.py** — основной модуль, содержащий реализацию алгоритма поиска мостов.
* **visualize.py** — модуль, отвечающий за построение и сохранение графических изображений графов.
* **test.py** — модуль с автоматическими тестами для проверки корректности алгоритма.
* **example.py** — файл с демонстрационным примером использования алгоритма.
* **requirements.txt** — список зависимостей, используемых в проекте.
* **README.md** — краткая текстовая инструкция по запуску и использованию проекта.

**Заключение**

В ходе выполнения курсовой работы была разработана и реализована программа для поиска мостов в неориентированном графе. Задача поиска мостов относится к числу важных в теории графов и находит широкое применение в прикладных задачах — от анализа устойчивости сетей до обнаружения критических связей в социальных или транспортных системах.

В работе был реализован эффективный алгоритм на основе поиска в глубину (DFS), обладающий линейной сложностью **O(V + E)**, что позволяет использовать его на графах большого размера. Реализация охватывает не только сам алгоритм, но и систему автоматического тестирования, визуализацию результата и пример использования. Такой подход позволяет не только убедиться в корректности работы программы, но и удобно демонстрировать её результат.

Проект обладает рядом преимуществ:

* Простая и понятная структура;
* Отделение логики, визуализации и тестов по отдельным модулям;
* Возможность легко модифицировать или расширять функциональность;
* Автоматизированные тесты повышают надёжность при доработке;
* Возможность визуально анализировать найденные мосты.

Разработанная программа может быть использована как основа для более сложных решений, например:

* анализа сетевой устойчивости;
* динамического обновления графа и мостов в реальном времени;
* работы с взвешенными графами или ориентированными структурами;
* интеграции с пользовательским интерфейсом или веб-приложением.

Таким образом, поставленная задача была успешно решена, и полученное решение может быть эффективно применено в различных задачах анализа графов как в учебных, так и в прикладных целях.

* анализа сетевой устойчивости;
* динамического обновления графа и мостов в реальном времени;
* работы с взвешенными графами или ориентированными структурами;
* интеграции с пользовательским интерфейсом или веб-приложением.

Таким образом, поставленная задача была успешно решена, и полученное решение может быть эффективно применено в различных задачах анализа графов как в учебных, так и в прикладных целях.

**Источник**

Исходный код - <https://github.com/Peshkov-Matvei/bridge-finder>.