|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
| МИНОБРНАУКИ РОССИИ | | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | | | |
| **Институт искусственного интеллекта** | | |  |
| **Кафедра высшей математики** | | |  |

|  |  |
| --- | --- |
| **КУРСОВАЯ РАБОТА** | |
| **по дисциплине** | |
| **«**Алгоритмы и теория сложности**»** | |
| **Тема курсовой работы**  **«Проверка графа на двудольность»** | |
| Студент группы КМБО- |  |
| Руководитель курсовой работы | *Драгилева И.П.* |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2024 г. | *(подпись студента)* |
|  |  |  |
| «Допущен(ы) к защите» | «\_\_»\_\_\_\_\_\_\_2024 г. | *(подпись руководителя)* |

Москва – 2024

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | | | | **Институт искусственного интеллекта** | | | | **Кафедра высшей математики** | | | | | | |  |
|  | | **Утверждаю** | | |
|  | | Заведующий  кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_Шатина А.В. | | |
|  | | «\_2\_» \_декабря\_\_2024г. | | |
| **ЗАДАНИЕ** | | | | |
| **на выполнение курсовой работы** | | | | |
| **по** **дисциплине** «Алгоритмы и теория сложности» | | | | |
|  | | | | |
| Студент Группа *КМБО-.* | | | | |
|  | | | | |
| 1. **Тема: «Проверка графа на двудольность»** | | | | |
| **2. Исходные данные:** тестовые примеры и наборы данных для испытаний | | | | |
| **3**. **Перечень вопросов, подлежащих обработке, и обязательного графического материала:**  1) Алгоритм(ы) решения задачи на языке высокого уровня.  2) Получение и сравнение количественных результатов при различных исходных данных.  3) Асимптотические оценки временной сложности. | | | | |
|  | | | | |
| **4. Срок представления к защите курсовой работы:** **до** «20» декабря 2024 г. | | | | |
|  | | | | |
| Задание на курсовую  работу выдал | «1» октября 2024г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | (*\_\_\_\_\_\_\_\_\_\_\_\_*) | |
| Задание на курсовую  работу получил | «\_\_\_»\_\_\_\_\_\_2024г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | (\_\_\_\_\_\_\_\_\_\_\_\_) | |

**Оглавление**

[**Введение** 4](#_Toc183261337)

[**Постановка задачи** 5](#_Toc183261338)

[**Описание алгоритмов** 6](#_Toc183261339)

[**Доказательство корректности** 11](#_Toc183261340)

[**Асимптотическая оценка временной сложности алгоритмов** 13](#_Toc183261341)

[**Результаты запуска программы** 14](#_Toc183261342)

[**Заключение** 15](#_Toc183261343)

[**Список литературы** 16](#_Toc183261344)

[**Приложения** 17](#_Toc183261345)

## **Введение**

Проверка графа на двудольность является одной из фундаментальных задач теории графов, которая находит применение в различных областях науки и техники.

Граф называется **двудольным**, если его вершины можно разбить на два множества так, чтобы все рёбра соединяли вершины из разных множеств.

Это свойство позволяет решать многие задачи, связанные с оптимизацией, разбиением данных и поиском соответствий. В современном мире алгоритмы проверки графа на двудольность имеют широкое применение:

* Взаимодействие между пользователями и серверами в сетях связи.
* Разработка алгоритмов маршрутизации в транспортных системах.
* Создание рекомендательных систем, где двудольный граф моделирует взаимодействие между пользователями и товарами (например, пользователь — фильм, пользователь — товар).
* Задачи составления расписаний, где вершины представляют задачи и временные слоты, а рёбра — допустимые назначения и т.д.

Алгоритмы проверки графа на двудольность на основе **поиска в глубину** **(DFS)** и **поиска в ширину** **(BFS)** являются простыми, но мощными инструментами, которые обеспечивают быстрое и эффективное решение задачи. В данном отчёте рассматривается реализация этих методов с применением к реальным данным, а также анализ полученных результатов.

## **Постановка задачи**

*Дано:*

Неориентированный граф задан в виде списка рёбер, где каждая строка описывает связь между двумя вершинами (представляет одно ребро в формате:

*Найти:* является ли заданный граф **двудольным** (т.е. можно ли разбить его вершины на два множества так, чтобы все рёбра соединяли вершины из разных множеств).

*Необходимо реализовать два алгоритма проверки двудольности* (алгоритмы должны корректно работать как для связных, так и для несвязных графов).

* Метод на основе поиска в ширину (BFS).
* Метод на основе поиска в глубину (DFS).

*Проверить решение задачи* для двух наборах данных:

* Тестовый пример (кол-во вершин: 12, кол-во ребер: 20)
* Усложненный набор данных (кол-во вершин: 875714)
* Созданный набор данных, где граф является двудольным (кол-во вершин: 12)

Также *необходимо оценить время выполнения алгоритма* и привести *асимптотическую оценку* временной сложности разработанного решения, обосновав её.

## **Описание алгоритмов**

* 1. ***Алгоритм проверки графа на двудольность поиском в ширину (BFS).***

Алгоритм проверки двудольности с использованием BFS поочерёдно обходит вершины графа, окрашивая их в два разных цвета, что соответствует двум разным множествам, и проверяет, чтобы никакие соседние вершины не имели одинакового цвета.

*Описание работы алгоритма:*

1. *Инициализация:*

* Создаём словарь для хранения цветов вершин. Цвет каждой вершины может быть либо 0, либо 1. Неокрашенные вершины отсутствуют в словаре.
* Для проверки всех компонент графа и учёта несвязных графов перебираем все вершины. Если вершина ещё не окрашена, начинаем с неё новый процесс BFS.

1. *Начало BFS:*

* Для текущей стартовой вершины назначаем начальный цвет (например, 0) и добавляем её в очередь.
* Используем очередь для поочерёдной обработки всех вершин графа.

1. *Обход соседей:*

* Извлекаем вершину из очереди и проверяем всех её соседей.
* Если соседняя вершина ещё не окрашена, окрашиваем её в противоположный цвет и добавляем в очередь для дальнейшей обработки.
* Если соседняя вершина уже окрашена в тот же цвет, что и текущая, то граф не является двудольным. Алгоритм завершается с отрицательным результатом.

1. *Проверка остальных компонент:* если BFS успешно завершился для текущей компоненты, продолжаем обработку остальных вершин графа, начиная BFS для каждой неокрашенной вершины.
2. *Завершение:* если все вершины были успешно обработаны без конфликтов, граф является двудольным.

|  |
| --- |
| ***Is\_bibartite\_BFS***  ***Вход:*** словарь смежности (graph): каждая вершина — ключ, а его значение — список соседей.  ***Выход:*** булево значение (True – данный граф является двудольным, False – данный граф не является двудольным) |
| 1. color = {} – пустой словарь для хранения цветов вершин  2. :  3. :  4. создать очередь (queue) и добавить в неё вершину node  5. (присваиваем первый цвет)  6. пока  7. current = (извлечь из очереди вершину current)  8. for neighbor in :  9. :  10.  11. добавить соседей neighbor в очередь  12. :  13. вернуть (граф не двудольный)  14. вернуть (граф двудольный) |

**Псевдокод 1.** Псевдокод алгоритма проверки двудольности с использованием BFS.

Рассмотрим на тестовом примере выполнение алгоритма проверки графа на двудольность поиском в ширину.

Представим для удобства наш тестовый набор данных в виде словаря, где вершина является ключом, а значение каждой вершины – список его соседей. В таблице 1 показано, как исходные данные были преобразованы в словарь

|  |  |
| --- | --- |
| Исходные данные из тестового набора данных | Преобразованные в словарь данные из тестового файла |
| 1 2  2 3  2 4  2 5  3 6  4 5  4 7  5 2  5 6  5 7  6 3  6 8  7 8  7 10  8 7  9 7  10 9  10 11  11 12  12 10 | {1: [2],  2: [1, 3, 4, 5],  3: [2, 6],  4: [2, 5, 7],  5: [2, 4, 6, 7],  6: [8, 3, 5],  7: [4, 5, 8, 9, 10],  8: [6, 7],  9: [10, 7],  10: [9, 11, 12, 7],  11: [10, 12],  12: [10, 11]} |

***Таблица 1.*** Преобразование исходных данных в словарь на примере тестового набора данных.

Приступим к выполнению нашего алгоритма:

1. *Инициализация:*

Стартуем с пустого словаря цветов:

Граф в виде списка смежности представлен на таблице 1.

1. ***Процесс обхода (BFS):***
2. Начинаем с вершины 1. Она ещё не окрашена, поэтому назначаем ей цвет 0 и добавляем вершину в очередь:
3. Извлекаем вершину 1 из очереди и смотрим на её соседа — вершину 2. Вершина 2 ещё не окрашена, назначаем ей цвет 1:

Добавляем вершину 2 в очередь.

1. Извлекаем вершину 2 из очереди и смотрим на её соседей:

Вершина 1 уже окрашена в цвет 0 (противоположный), поэтому продолжаем.

Вершина 3 не окрашена, назначаем ей цвет 0:

Добавляем вершину 3 в очередь.

Вершина 4 не окрашена, назначаем ей цвет 0:

Добавляем вершину 4 в очередь.

Вершина 5 не окрашена, назначаем ей цвет 0:

Добавляем вершину 5 в очередь.

1. Извлекаем вершину 3 из очереди и смотрим на её соседей:

Вершина 2 уже окрашена в цвет 1 (противоположный), продолжаем.

Вершина 6 не окрашена, назначаем ей цвет 1:

Добавляем вершину 6 в очередь.

1. Извлекаем вершину 4 из очереди и смотрим на её соседей:

Вершина 2 уже окрашена в цвет 1 (противоположный), продолжаем.

Вершина 5 уже окрашена в цвет 0, где **мы сталкиваемся с конфликтом**, и алгоритм должен вернуть— **граф не двудольный**.

* 1. ***Алгоритм проверки графа на двудольность поиском в глубину (DFS).***

Концепция данного алгоритма очень похожа на концепцию алгоритма проверки графа на двудольность в ширину. Многим данный алгоритм нравится больше, так как является чуть-чуть более простым и не требующим дополнительных структур данных, в отличие от BFS, где используется очередь.

*Описание работы алгоритма:*

1. Инициализация:

* Как и в алгоритме BFS, каждая вершина начинает без цвета (неокрашена).
* Мы будем использовать два цвета (например, 0 и 1), чтобы раскрасить вершины.

1. Поиск в глубину (DFS):

* Для каждой вершины, которая ещё не была посещена, мы начинаем выполнение DFS.
* В процессе обхода вершины присваиваются цвета. Если вершина уже окрашена, мы проверяем, что все её соседи окрашены в противоположный цвет.
* Если мы находим соседей с одинаковым цветом, то **граф не двудольный,** и алгоритм завершает выполнение.

*Какое отличие от BFS?*

DFS (поиск в глубину) использует рекурсию (или стек), чтобы углубиться в граф до конца пути перед возвращением и продолжением обхода, в отличие от BFS (поиск в ширину), который использует очередь для обхода графа по уровням, начиная с исходной вершины, и исследует все соседние вершины на одном уровне перед тем, как перейти к следующему уровню.

|  |
| --- |
| ***Is\_bibartite\_DFS***  ***Вход:*** словарь смежности (graph): каждая вершина — ключ, а его значение — список соседей.  ***Выход:*** булево значение (True – данный граф является двудольным, False – данный граф не является двудольным) |
| :  1. - пустой словарь для хранения цветов вершин  2. :  3.  4. if – рекурсивно вызываем функцию обратную DFS для node с цветом 0  5. вернуть *False* (граф не двудольный)  6. Вернуть *True* (граф двудольный)  :  1.  2.  3.  4.  5.  6. вернуть *False* (граф не двудольный)  7. вернуть *True* (успешно раскрасили вершину node и её соседей) |

**Псевдокод 2.** Псевдокод алгоритма проверки двудольности с использованием DFS.

Посмотрим, чем отличается алгоритм проверки двудольности с использованием DFS от BFS на тестовом примере:

1. *Инициализация:*

Сначала создаем пустой словарь color, чтобы хранить цвет каждой вершины.

Мы начинаем обход с вершины 1 и присваиваем ей цвет 0.

1. ***Процесс обхода (DFS):***
2. Вершина 1:

Присваиваем цвет 0.

Смотрим на соседей: вершина 2. Поскольку 2 не окрашена, вызываем DFS для вершины 2 с цветом 1.

1. Вершина 2:

Присваиваем цвет 1.

Смотрим на соседей: вершины 1, 3, 4, 5.

Вершина 1 уже окрашена в цвет 0 (противоположный), продолжаем.

Вершина 3 не окрашена, вызываем DFS для вершины 3 с цветом 0.

1. Вершина 3:

Присваиваем цвет 0.

Смотрим на соседей: вершины 2, 6.

Вершина 2 уже окрашена в цвет 1 (противоположный), продолжаем.

Вершина 6 не окрашена, вызываем DFS для вершины 6 с цветом 1.

1. Вершина 6:

Присваиваем цвет 1.

Смотрим на соседей: вершины 3, 5, 8.

Вершина 3 уже окрашена в цвет 0 (противоположный), продолжаем.

Вершина 5 не окрашена, вызываем DFS для вершины 5 с цветом 0.

1. Вершина 5:

Присваиваем цвет 0.

Смотрим на соседей: вершины 2, 4, 6, 7.

Вершина 2 уже окрашена в цвет 1 (противоположный), продолжаем.

Вершина 4 не окрашена, вызываем DFS для вершины 4 с цветом 1.

1. Вершина 4:

Присваиваем цвет 1.

Смотрим на соседей: вершины 2, 5, 7.

Вершина 2 уже окрашена в цвет 1, где **мы сталкиваемся с конфликтом**, и алгоритм должен вернуть— **граф не двудольный**.

## **Доказательство корректности**

* 1. ***Алгоритм проверки графа на двудольность поиском в ширину (BFS).***

Для доказательства корректности данного алгоритма *параметризуем двудольность*:

Граф является **двудольным**, если его вершины могут быть разделены на два подмножества 𝐴 и 𝐵, так что рёбра соединяют только вершины из разных подмножеств, то есть для всех рёбер , , либо наоборот.

*Рассмотрим свойства алгоритма BFS*:

В алгоритме *BFS* используется очередь для обхода графа. Мы начинаем с вершины и присваиваем ей цвет 0 (например). Для всех её соседей , , …, которые еще не окрашены, мы присваиваем противоположный цвет (1). После этого BFS продолжает обходить граф. Для каждого соседа, если он ещё не окрашен, он получает противоположный цвет.

Если на каком-либо шаге алгоритм встречает соседей, которые имеют тот же цвет, что и текущая вершина, это **нарушает условие двудольности**, и алгоритм возвращает **False**.

***Корректность окраски:*** если вершины и соединены ребром, то при выполнении алгоритма BFS они будут окрашены в разные цвета. Это свойство сохраняется на протяжении всего алгоритма, так как при обходе по уровням все вершины одного уровня получают один цвет, а все вершины следующего уровня — противоположный.

***Отсутствие конфликтов:*** если при обработке соседей мы находим, что сосед уже окрашен в тот же цвет, что и текущая вершина, это означает, что эти *вершины принадлежат к одному подмножеству и имеют общую связь*, то есть **граф не является двудольным**. В этом случае алгоритм **немедленно возвращает False**.

***Обход всех рёбер:*** Алгоритм проходит все рёбра графа. Если ни одно из рёбер не нарушает правила двудольности, значит, граф двудольный.

* 1. ***Алгоритм проверки графа на двудольность поиском в глубину (DFS).***

Доказательство корректности алгоритма случае поиска в глубину то же, что и в ширину. Остается рассмотреть только *свойства алгоритма DFS*:

Алгоритм DFS выполняет рекурсивный обход графа, в отличие от BFS, начиная с неокрашенной вершины и присваивая ей цвет (например, 0). Все соседние вершины получают противоположный цвет (например, 1). Затем для каждого соседа, который ещё не окрашен, рекурсивно вызывается DFS с противоположным цветом. Этот процесс продолжается до тех пор, пока не будут обработаны все вершины или не будет обнаружен конфликт.

***Корректность окраски****:* на каждом шаге алгоритм присваивает соседям текущей вершины противоположный цвет. Если на каком-либо этапе алгоритм пытается присвоить соседям тот же цвет, что и текущей вершине, это приводит к конфликту, который говорит о том, что граф не двудольный. Таким образом, алгоритм правильно окрашивает граф.

***Отсутствие конфликтов****:* если граф двудольный, то при любом обходе DFS не будет конфликтов, так как все вершины будут окрашены в противоположные цвета на основе их уровней. Конфликт возникает только в случае, если граф не двудольный, то есть если существует рёбер, соединяющих вершины одного и того же подмножества.

***Обход всех рёбер****:* алгоритм *DFS* проходит по всем рёбрам графа, так как в цикле мы указываем обход **для каждой вершины**. Если ни одно из рёбер не нарушает правила двудольности, то алгоритм завершится, не обнаружив конфликтов, и вернёт *True*.

## **Асимптотическая оценка временной сложности алгоритмов**

* 1. ***Алгоритм проверки графа на двудольность поиском в ширину (BFS).***

Общее время, затраченное на обработку всех вершин и рёбер, пропорционально где:

* — количество вершин графа.
* — количество рёбер графа.

Это объясняется тем, что каждая вершина будет посещена не более одного раза, а каждое ребро будет обработано не более одного раза.

***Доказательство:***

* В процессе *BFS* мы начинаем с некоторой вершины, помещаем её в очередь и начинаем её обработку. Для каждого соседа этой вершины мы проверяем его цвет и помещаем в очередь, если он ещё не был посещён.
* Каждый элемент в очереди будет посещён не более одного раза, так как мы помечаем вершины как посещённые.
* Поскольку обработка каждой вершины занимает времени (окрашивание и проверка соседей), а обработка каждого рёбра тоже (проверка на окраску соседей), время выполнения алгоритма пропорционально сумме числа вершин и рёбер.
  1. ***Алгоритм проверки графа на двудольность поиском в глубину (DFS).***

Общее время алгоритма, использующего *DFS*, затраченное на обработку всех вершин и рёбер, пропорционально как и в случае с алгоритмом, использующим *BFS*, где:

* — количество вершин графа.
* — количество рёбер графа.

***Доказательство:***

* В процессе *DFS* каждая вершина посещается ровно один раз, и для каждого ребра выполняется проверка на его соседей.
* Если использовать рекурсию, то в каждом рекурсивном вызове мы обрабатываем вершину и её соседей. Для каждого вызова рекурсии выполняется одна операция для вершины, и затем мы рекурсивно вызываем *DFS* для каждого соседа.
* Как и в *BFS*, обработка каждой вершины и каждого рёбра требует времени.

## **Результаты запуска программы**

Приступим к тестированию наших алгоритмов на предложенных наборах данных. Напомню, что у нас есть 2 набора данных:

* ***Тестовый пример*** (кол-во вершин: 12, кол-во ребер: 20)
* ***Усложненный набор данных*** (кол-во вершин: 875714)
* ***Созданный набор данных, где граф является двудольным*** (кол-во вершин: 12)

Каждый файл описывает экземпляр задачи об определении двудольности графа и имеет формат *списка ребер без повторений*:

[вершина\_1 (откуда выходит ребро)] [вершина\_2 (куда приходит ребро)]

[вершина\_3 (откуда выходит ребро)] [вершина\_4 (куда приходит ребро)]

…

Для начала я решил упростить себе задачу и привести исходные данные к формату списка смежности. Для этого я написал функцию , которая преобразует список рёбер графа в **список смежности** — структуру данных, которая удобно описывает граф в виде словаря, где ключи — вершины графа, а значения — списки соседних вершин.

Описание функции :

1. Создаётся объект *defaultdict* с типом значений *set*. Это позволяет автоматически добавлять новые вершины без необходимости вручную проверять их наличие.
2. Далее идет заполнение списка смежности. Мы проходимся по всем ребрам вершина 𝑣 добавляется в множество соседей вершины 𝑢; вершина 𝑢 добавляется в множество соседей вершины 𝑣 (так как граф неориентированный).
3. Для удобства использования множества соседей (тип *set*) преобразуются в списки (тип *list*).

В таблице 2 приведено сравнение двух алгоритмов проверки графов на двудольность (с обходом графа в ширину и глубину) по времени выполнения и результатами на обоих наборах данных.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Алгоритм | Время выполнения на тестовом наборе данных | Время выполнения на усложненном наборе данных | Время выполнения на созданном наборе данных, где граф явл. двудольным | Результат на тестовом наборе данных | Результат на усложненном наборе данных | Результат на созданном наборе данных, где граф явл. двудольным |
| BFS | < 0.00001 с. | 0.0020041465759277344 секунд | < 0.00001 с. | False | False | True |
| DFS | < 0.00001 с. | 0.0003273487091064453 секунд | < 0.00001 с. | False | False | True |

**Таблица 2.** Сравнение алгоритмов проверки графов на двудольность (с обходом графа в ширину и глубину).

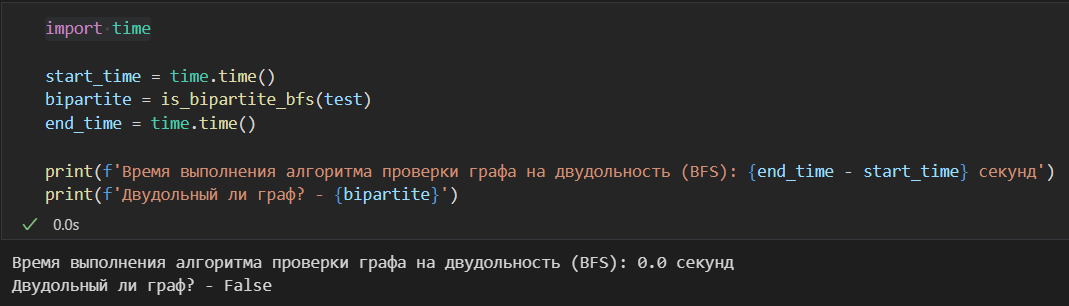
## **Заключение**

В ходе курсовой работы были рассмотрены **алгоритмы проверки графа на двудольность** с помощью разных обходов в графе. Хотя оба алгоритма имеют **одинаковую** **асимптотическую оценку**  где — количество вершин графа; — количество рёбер графа, **алгоритм, базирующийся на обходе графа в глубину**, оказался продуктивней, выполнив задачу за 0.0003273487091064453 секунды в усложненном наборе данных, превзойдя **алгоритм, базирующийся на обходе графа в ширину**, в более чем в 6 раз.

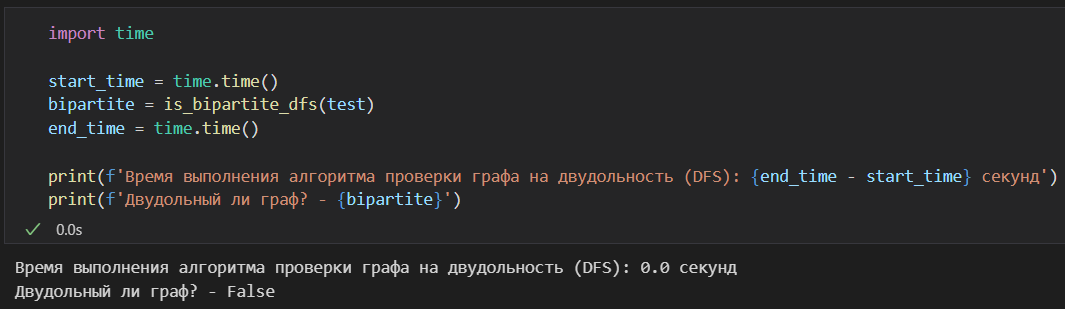
## **Список литературы**

1. Харари Ф. Теория графов. /пер. с англ. — изд. 2-е — М.: Едиториал УРСС, 2003. — 296 с. — ISBN 5-354-00301-6
2. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - 3-e изд. - СПб: Питер, 2024. - 288 с.
3. Рафгарден Т. Совершенный алгоритм. Основы. - 2-e изд. - СПб: Питер, 2023. - 256 с.
4. Скиена С.С. Алгоритмы. Руководство по разработке. - 2-е изд. - М.: БХБ, 2018. - 720 с.

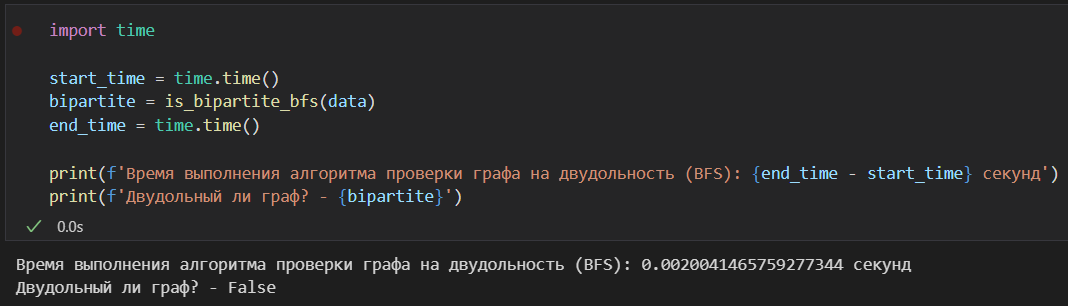
## **Приложения**



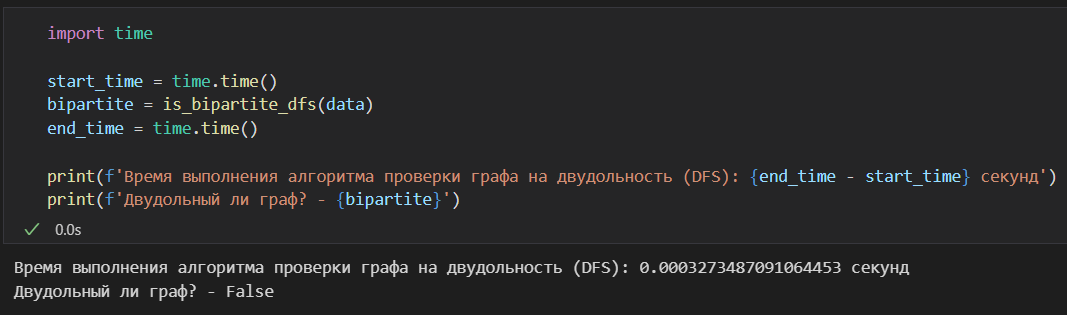
***Рисунок 1.*** Результат выполнения алгоритма проверки данных на двудольность поиском в ширину на тестовых данных.



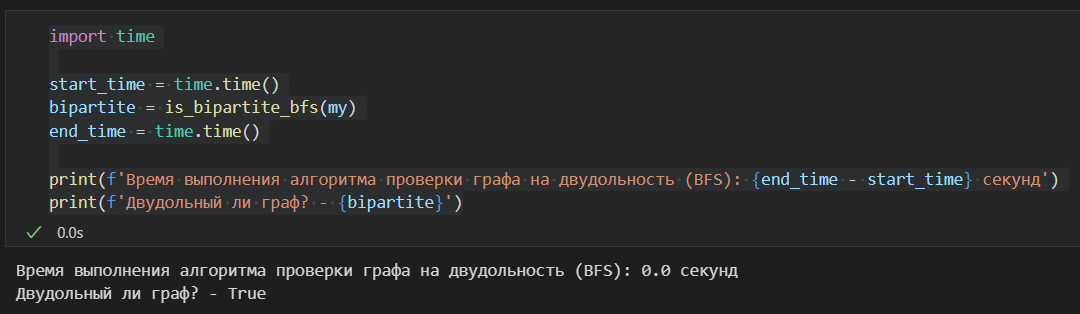
***Рисунок 2.*** Результат выполнения алгоритма проверки данных на двудольность поиском в глубину на тестовых данных.



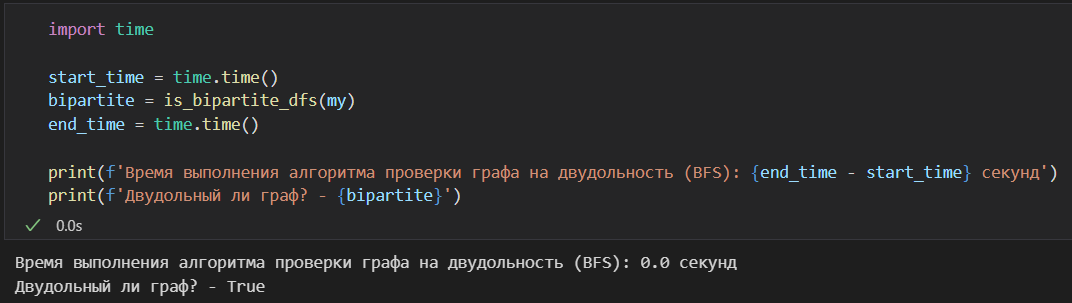
***Рисунок 3.*** Результат выполнения алгоритма проверки данных на двудольность поиском в ширину на усложненных данных.



***Рисунок 4.*** Результат выполнения алгоритма проверки данных на двудольность поиском в глубину на усложненных данных.



***Рисунок 5.*** Результат выполнения алгоритма проверки данных на двудольность поиском в ширину на созданном наборе данных, где граф является двудольным.



***Рисунок 6.*** Результат выполнения алгоритма проверки данных на двудольность поиском в глубину на созданном наборе данных, где граф является двудольным.