Санкт-Петербургский политехнический университет Петра Великого

Кафедра прикладной математики и информатики

**Дискретная математика**

Отчет по лабораторной работе № 3

Тема "Деревья"

**Вариант 2 – «Проверка свойства древочисленности (субцикличность)»**

Выполнил студент гр. 5030102/20202 Тишковец С. Е.

Санкт-Петербург

2024

**Оглавление**

[**Постановка задачи** 3](#_Toc179830351)

[**Язык программирования** 3](#_Toc179830352)

[**Описание алгоритма** 3](#_Toc179830353)

[**Демонстрация алгоритма** 5](#_Toc179830354)

[**Формат данных** 6](#_Toc179830355)

[**Дополнительные требования** 7](#_Toc179830356)

[**Область применимости алгоритма** 7](#_Toc179830357)

[**Источники информации** 8](#_Toc179830358)

# ****Постановка задачи****

*Вариант 2 - Проверка свойства древочисленности (субцикличность)*

На вход программе подаётся граф.

Проверить, является ли граф деревом, используя утверждение 7 из теоремы параграфа “Основные свойства свободных деревьев”. Если нет, то в выходной файл написать, что нарушено: ацикличность (в этом случае предоставить хотя бы один найденный цикл), субцикличность (в этом случае указать для какого ребра это неверно) или и то, и другое. В любом случае проверить, является ли граф древочисленным.

По итогу нужно запустить программу на различных графах и увидеть, что либо граф является деревом и древочисленный, либо то, что если ровно одно из условий из утверждения 7 не выполнено, то граф не будет древочисленным, либо то, что если оба условия не выполнены, то граф как может, так и не может быть древочисленным (всё перечисленное должно быть верно за исключениями из утверждения 6).

Здесь используются утверждения 5, 6 и 7 из теоремы параграфа “Основные свойства свободных деревьев”.

Дополнительные требования:

* указать сложность алгоритма и доказать, что она именно такая;
* объяснить почему был выбран тот или иной способ представления графов в программе.

# ****Язык программирования****

Алгоритм был реализован с помощью **C++ 14**

# ****Описание алгоритма****

**Проверка графа на ацикличность:**

**Для проверки графа на ацикличность используются 2 функции: 1-я находит все циклы в графе, начиная с заданной вершины; 2-я находит все циклы для всех вершин графа.**

* 1. **Функция для нахождения всех циклов в графе *findCyclesFromNode,* начиная с заданной вершины (всего вершин *V*), использует поиск в глубину (DFS). Создается стек *s* для хранения пар (вершина, родитель) и массив *parentArray* для хранения родителей.**
  2. **Запускается цикл до тех пор, пока стек не опустеет. На каждой итерации из стека извлекается верхняя пара вершин.**
  3. **Далее проверяется, посещена ли текущая вершина, а также все ее соседи. Если нет, помечаем вершину как посещённую, а соседей добавляем в стек для дальнейшей обработки. Записываем текущую вершину как родителя соседей.**
  4. **Если сосед посещён и не является родительской вершиной, это означает, что мы нашли простой цикл. Для восстановления цикла мы начинаем с текущей вершины и движемся к родителям по массиву *parentArray*, добавляя их в вектор *cycle* до тех пор, пока не вернемся к найденному соседу, который и является одной из вершин цикла. После завершения цикла добавляем текущую вершину в *cycle,* чтобы сформировать запись цикла в формате, когда начальная и исходная вершины совпадают. Затем сохраняем цикл в набор всех циклов *Allcycles* в графе.**
  5. **В функции для нахождения всех циклов у всех вершин графа *findAllCycles* создается вектор посещенных вершин *visited[]* и инициализируется значениями *false* – это свидетельствует о том, что на начальном этапе все вершины не посещены. Благодаря вектору*****visited[]* мы не будем проверять одну и ту же вершину дважды.**
  6. **Далее функция проходит по всем вершинам графа. Если текущая вершина *v* ещё не была посещена, вызывается функция *findCyclesFromNode* с этой вершиной. В результате работы алгоритма у нас найдены и записаны все простые циклы, содержащиеся в графе.**

**Проверка графа на субцикличность:**

1. **Функция *checkSubcyclicity* должна проверить, можно ли добавить ребро между всеми парами несмежных вершин графа, чтобы в результате образовался ровно один простой цикл.**
2. **Запускается 2 вложенных цикла, каждый перебирает *V* значений. Первый цикл по*****u* перебирает все вершины графа, начиная с первой. Второй цикл по*****v* также перебирает все вершины графа и используется для того, чтобы проверить, является ли вершина *v* соседом *u.***
3. **Если алгоритм обнаружил две несмежные вершины, в граф добавляется временное ребро между данными вершинами, чтобы проверить, выполнится ли условие субцикличности. Вызывается функция *findAllCycles*, которая находит все циклы в графе с учетом нового добавленного ребра и сохраняет их в *allCycles.***
4. **Временное ребро удаляется и далее идет проверка условия субцикличности: если количество найденных простых циклов не равно единице, тогда это означает, что граф не является субцикличным.**

**Проверка графа на древочисленность:**

1. **Подсчитывается количество ребер в графе. Если количество ребер на единицу меньше, чем количество вершин, проверка успешна и граф является древочисленным.**

# ****Демонстрация алгоритма****

Изображение выглядит как диаграмма, зарисовка, линия, круг

Автоматически созданное описаниеПример 1. Рассмотрим следующий граф:

Записать граф можно в виде списка пар ребер между вершинами:

**(0, 1)**

**(1, 2)**

**(2, 3)**

**(1, 3)**

**Шаг 1: Проверка ацикличности.**

**Начиная с вершины 0, перебираем все вершины графа и их соседей, запоминаем у какой вершины какой родитель. Массив *parentArray* на начальном этапе инициализируется значениями -1, что означает отсутствие родителя. У начальной вершины 0 нет родителя, поэтому в массиве *parentArray* первое значение всегда будет -1. Общий вид массива *parentArray = { -1, 0, 1, 2 },* где на позиции с индексом *i* стоит номер вершины, которая является родителем для вершины *i.***

**Если оказывается так, что сосед посещён и не является родителем, то найден цикл. В нашем примере вершина 1 является соседом для вершины 3, однако вершина 1 не является родителем для вершины 3. Получаем цикл 1 -> 2 -> 3 -> 1. Следовательно, граф не является ацикличным. Записываем цикл в файл.**

**Шаг 2: Проверка субцикличности.**

**Начиная с вершины 0, перебираем все вершины графа и находим такие пары вершин, которые не являются смежными. Первая такая пара вершин: 0 и 2. Добавляем временное ребро *(0, 2)*, соединяющее эти 2 вершины – получим следующий граф:**

Изображение выглядит как зарисовка, рисунок, диаграмма, круг

Автоматически созданное описание

**Далее вызываем функцию для нахождения всех циклов в графе. Результат работы функции *findAllCycles* – 3 простых цикла.**

**1 -> 2 -> 3 -> 1**

**0 -> 1 -> 2 -> 0**

**0 -> 1 -> 3 -> 2 -> 0**

**Таким образом, в графе нарушено условие субцикличности.**

**Удаляем временное ребро *(0, 2)* и записываем в файл вершины, для которых, собственно, проверка на субцикличность была неуспешной.**

**Шаг 3: Проверка древочисленности.**

**Для того, чтобы граф обладал свойством древочисленности, необходимо, чтобы количество вершин было на единицу больше, чем количество ребер. В нашем примере 4 вершины и 4 ребра, следовательно, граф не является древочисленным.**

# ****Формат данных****

Входные данные: список смежности считывается из текстового файла. В первой строке указано количество вершин в графе, далее построчно, начиная со второй строки, в файле записаны ребра графа, соединяющие 2 вершины. Формат записи ребра в строке: **x\_y**, где **x** и **y** – смежные вершины в графе, \_ - означает пробел. Запись одного и то же ребра дважды недопустима. Все символы текстового файла являются целыми числами. Граф должен быть задан в файле как неориентированный, поскольку дерево – это неориентированный граф, для которого соблюдаются условия ацикличности и субцикличности.

Выходные данные: результатом выполнения программы являются 3 сообщения о результатах проверки каждого из свойств. В случае нарушения свойства ацикличности в файл записываются все найденные простые циклы. В случае нарушения свойства субцикличности в файл записывается пара вершин, для которых проверка была неудачной. В случае нарушения свойства древочисленности в файл записывается количество ребер и количество вершин, на единицу меньше. Сообщения дублируются в консоль в таком же формате.

# ****Дополнительные требования****

* **Временная сложность для каждого из трех свойств:**

1. Ацикличность: , где V – количество вершин, E – количество рёбер. Для графа с E рёбрами и V вершинами внешний цикл ***while*** и вложенный цикл ***for*** за счет проверки с использованием массива ***visited[]*** обрабатывают каждое из Е рёбер и каждую из V вершин ровно один раз, давая сложность .
2. Субцикличность: , где V – количество вершин, E – количество рёбер. Для графов, где : . В функции запускается 2 вложенных цикла ***for***, каждый из которых перебирает V вершин. Получаем . В ходе выполнения алгоритма, во втором цикле (внутренний цикл ***for***) вызывается функция для поиска всех циклов в графе – ее временная сложность . В итоге для всего алгоритма получим сложность .
3. Древочисленность: , где V – количество вершин в графе. Для проверки этого свойства необходимо пройтись циклом ***for*** по всем вершинам графа, начиная с первой, и посчитать количество соседей у каждой из V вершин.

* **Способ представления графа** в данной программе – список смежности – был выбран не спроста. Список смежности занимает  памяти, что может быть значительно меньше (особенно если граф слабо связный), чем , требуемое для матрицы смежности, или для матрицы инцидентности. Также, обход соседей в списке смежности требует времени (для графов без кратных рёбер) в отличие от для матрицы смежности.

# ****Область применимости алгоритма****

1. Область применения:

**Данный алгоритм позволяет определить, является ли заданный граф деревом. Для того, чтобы граф был деревом, необходимо и достаточно, чтобы выполнялись 2 свойства – ацикличность и субцикличность. Также программа позволяет проверить граф на древочисленность.**

**Например, ацикличность требуется для нормализации и поддержания внутренней целостности данных в базах данных, использующих графовые технологии.**

**К примеру, в задачах оптимизации транспортных систем, таких как построение маршрутов доставки или связанное с логистикой проектирование, субцикличность используется для нахождения минимальной длины маршрута с ровно одним циклом.**

1. Случаи сбоя:

- Нарушение формата входных данных: например, если в файле не все элементы списка смежности (смежные вершины) будут заполнены, то алгоритм выдаст непредсказуемый результат.

- Если во входном файле какое-то ребро будет дублироваться, то программа неверно проверит граф на свойство древочисленности.

- Код работает только с неориентированными графами, при попытке проверить ориентированный граф на свойство, например, ацикличности алгоритм выдаст неверный результат.

# ****Источники информации****

* [https://disk.yandex.ru/i/g6JvWFBBmfvr0g](https://vk.com/away.php?to=https%3A%2F%2Fdisk.yandex.ru%2Fi%2Fg6JvWFBBmfvr0g&cc_key=)
* **https://studfile.net/preview/718680/page:7/**