Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и информатики

Лабораторная работа №3 по дисциплине

Дискретная математика

Тема: «Деревья»

Вариант 1 – Проверка свойства древочисленности (связность)

Выполнил студент гр.5030102/20202 Дрекалов Н. С.

Преподаватель Новиков Ф. А.

Санкт-Петербург

2024

Оглавление

[Формулировка задания 3](#_Toc183489148)

[Используемый язык программирования и системы сборки 3](#_Toc183489149)

[Описание алгоритма проверки: 3](#_Toc183489150)

[Проверка на связность и ацикличность 3](#_Toc183489151)

[Проверка на древочисленнность 4](#_Toc183489152)

[Демонстрация работы алгоритма 4](#_Toc183489153)

[Область применения реализованного алгоритма 7](#_Toc183489154)

[Формат входных и выходных данных 7](#_Toc183489155)

[Сложность алгоритма 8](#_Toc183489156)

[Сложность проверки на связность и ацикличность 8](#_Toc183489157)

[Cложность проверки на древочисленнность 8](#_Toc183489158)

[Общая сложность 8](#_Toc183489159)

[О способе представления графов в программе 9](#_Toc183489160)

[О связи древочисленности со связностью и ацикличностью 9](#_Toc183489161)

# Формулировка задания

В лабораторной работе требуется проверить, является ли неориентированный граф без петель деревом по определению, а также убедиться в выполнении свойств, связанных с древочисленностью графа

# Используемый язык программирования и системы сборки

* C++20
* CMake: 3.29.0
* Ninja: 1.12.1

# Описание алгоритма:

По определению дерево – это связный ациклический граф. Поэтому проверим, является ли граф связным и ациклическим. А затем отдельно проверим, является ли граф древочисленным.

## Проверка на связность и ацикличность

1. Создаём двумерный массив A – для хранения компонент связности, а также одномерный массив C для хранения цикла.
2. Создаем множество X и заполняем её всеми вершинами – это множество будет хранить в себе не посещенные вершины.
3. Выполняем пункты 4–9, пока множество вершин не пустое.
4. Создаём стек, в котором будем хранить: текущую вершину, вершину, из которой мы попали в текущую вершину (грубо говоря, родителя) и DFS путь до текущей вершины в виде списка.
5. Добавляем в массив A новую пустую строку.
6. Проходим DFS с помощью ранее созданного стека, начиная с первой вершины в множестве X – для каждой вершины в DFS выполняем пункты 7–9
7. Если вершина посещена (то есть её нет в X) – то это значит, что в программе есть цикл. Если до этого не был найден цикл (то есть C пустое) – присваиваем в C цикл, полученный из пути, которых хранился в стеке вместе с вершиной (для получения цикла из пути достаточно пройтись по этому пути в обратном порядке до того момента, пока в пути не найдётся текущая вершина)
8. Добавляем вершину в конец последней добавленной строки массива A и убираем вершину из множества X
9. Проходим циклом по всем смежным к текущей вершине вершинам и проверяем: если смежная вершина не совпадает с вершиной, из который мы пришли в текущую вершину, то добавляем смежную вершину в стек вместе с увеличенным DFS путём.
10. Получаем в множестве A компоненты связности – если число компонент равно 1, то граф связный. Также если C пустое, то граф ацикличный, иначе в C хранится один из циклов графа.

## Проверка на древочисленнность

1. Посчитаем все рёбра в графе – для этого заведём счётчик edgesCnt = 0
2. Перебираем все вершины в графе, и увеличиваем счётчик edgesCnt на количество смежных к текущей вершине вершин.
3. Таким образом, получаем в edgesCnt удвоенное количество рёбер (так как каждое ребро есть в 2-х вершинах). Соответственно, если edgesCnt / 2 на единицу меньше числа вершин, то граф древочисленный

# Демонстрация работы алгоритма

Для демонстрации работы алгоритма я проделаю его шаги для цикличного несвязного графа. A, C и X имеют такой же смысл, как и в описании алгоритма.

Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


Пояснения для пунктов 0–5 в виде картинок расположены после описания шагов

1. Начальное состояние – массивы A и C пустые, в X все вершины
2. Берем первую вершину из X – 1 – и кладём её на стек. Затем посещаем вершину 1 – добавляем её в массив A и убираем из X. У вершины 1 нету смежных вершин, поэтому на этом DFS от этой вершины заканчивается. X не пустой, а значит, надо повторить DFS
3. Берём первую вершину из X – 2 – и кладём её на стек. Затем аналогично вершине 1 посещаем её. Добавляем все смежные к вершине 2 вершины – 3 и 4 – на стек.
4. Берём вершину из стека – 4 – и посещаем её. Добавляем на стек смежную вершину 3 – вершину 2 не добавляем, так как из неё мы пришли на вершину 4.
5. Берём вершину из стека – 3 – и посещаем её. По аналогичным пункту рассуждениям на стек из смежных вершин нужно положить только 2.
6. Берём вершину из стека – 2 – она уже посещена. И массив C пустой – значит, надо в массив C переписать путь до вершины в обратном порядке 2 до момента, пока эта вершина не встретится ещё раз на пути. Получаем C = [2, 3, 4, 2] – весь путь в обратном порядке
7. На стеке осталась ещё вершина 3 – но после того, как мы её возьмем из стека, окажется, что она уже посещена и массив C не пустой – поэтому мы просто пропускаем её.
8. Получаем A= [[1], [2, 4, 3]] – количество строк больше 1, то есть граф несвязный. C = [2, 3, 4, 2] – не пустое, значит есть цикл.
9. Осталось проверить на древочисленность. Для этого проходим по всем вершинам, складываем количество инцидентных вершине рёбер, а затем это число делим на 2. Получаем число рёбер , то есть на единицу меньше количества вершин. Граф древочисленный.

Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


# Область применения реализованного алгоритма

Алгоритм можно применить для любого графа, в котором нету петлей. Алгоритм может быть полезен для относительно быстрого определения того, является ли граф деревом

*Случаи некорректной работы программы:*

* Пользователь указал путь к пустому файлу.
* Пользователь указал путь к файлу, которого не существует или у программы недостаточно прав для чтения из указанного места.
* Пользователь указал путь к файлу, который был поврежден или не содержит валидного представления графа.
* Пользователь указал путь к файлу, в котором записан граф с петлями.

В остальных случаях программа будет работать корректно.

# Формат входных и выходных данных

*Входные данные* – граф в файле, представленный как список смежности.

*Выходные данные* – информация о связности, ацикличности и древочисленности этого графа в файле. Также, если граф несвязный, в файл будут записаны компоненты связности, если граф ацикличный – будет записан цикл, если граф не древочисленный – будет записано число вершин и рёбер в этом графе

# Сложность алгоритма

Пусть – число вершин, а – число рёбер. A, C и X имеют такой же смысл, как и в описании алгоритма

## Сложность проверки на связность и ацикличность

1. Создание массивов A и C выполняется за
2. Для заполнения множества X используется цикл for по вершинам – сложность
3. Копирование данных в C при наличии цикла в графе займёт ) (худший случай – копирование V вершин – будет, если цикл в графе содержит все вершины).
4. Алгоритм обхода по всем вершинам хоть и представляет из себя 3 вложенных цикла, на самом деле имеет сложность обычного DFS – так как наиболее вложенный цикл – цикл перебора смежных вершин – выполняется для вершины n в худшем случае раз, где – число рёбер, инцидентных вершине n. При этом этот цикл будет вызван один раз для каждой вершины – то есть общая сложность обхода
5. 1, 2 и 4 выполняются последовательно, а 3 выполняется максимум один раз за алгоритм => общая сложность проверки

## Cложность проверки на древочисленнность

1. Для расчёта удвоенной суммы рёбер нужно один раз в цикле пройти по всем вершинам => сложность проверки на древочисленность

## Общая сложность

Так как обе проверки выполняются независимо друг от друга, **общая сложность алгоритма будет равна**

# О способе представления графов в программе

В программе граф представлен как массив смежности, где индексу i соответствует i-я вершина, и по индексу хранится массив смежных вершин.

Я выбрал этот способ из-за его в эффективности по памяти и быстроте нужных операций: граф занимает O ( памяти, где V – число вершин, а – число рёбер, при этом все нужные операции и алгоритмы выполняются за хорошую сложность:

* Получение списка смежных вершин –
* Перебор всех смежных к вершине вершин – где – число инцидентных вершине рёбер.
* DFS –

# О связи древочисленности со связностью и ацикличностью

Требовалось проверить с помощью программы связь древочисленности с ацикличностью и связностью. Для этого я создал 4–6 тестовых графов на каждую из возможных комбинаций значений свойств ацикличности и связности. Получилось, что

* Если граф ацикличный и связный, то он древочисленный
* Если граф имеет цикл и связный, то он не может быть древочисленным
* Если граф ацикличный и не связный, то он не может быть древочисленным
* Если граф ацикличный и не связный, то некоторые из тестовых графов были древочисленными, а некоторые – нет

Полученные результаты согласуются с теорией и ожиданиями