**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Графы, деревья»**

**Студент гр. 22Б16-пу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Олизько С.С.**

**Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дик А.Г.**

**Санкт-Петербург**

**2023 г**

Оглавление

[**Цель работы:** 3](#_Toc153215487)

[**Задача:** 3](#_Toc153215488)

[**Теоритическая часть:** 3](#_Toc153215489)

[**Алгоритм метода:** 5](#_Toc153215490)

[**Описание программы:** 6](#_Toc153215491)

[**Рекомендации для пользователя:** 6](#_Toc153215492)

[**Контрольный пример:** 8](#_Toc153215493)

[**Анализ результатов работы алгоритма и вводных условий:** 8](#_Toc153215494)

[**Вывод:** 10](#_Toc153215495)

[**Источники:** 10](#_Toc153215496)

# **Цель работы:**

С помощью алгоритмов работы с графами в виде деревьев, найти все возможные обходы дерева по узлам.

# **Задачи:**

Формализовать задачу, предоставленную в текстовом виде. Задать начальные условия. Найти все возможные обходы дерева.

# **Теоретическая часть:**

Для решения данной задачи подходит структура данных — очередь.

Очередь - это структура данных, которая работает по принципу "первым пришел, первым вышел" (First-In-First-Out, FIFO). Это означает, что элементы добавляются в конец очереди и извлекаются из начала очереди. Очередь можно сравнить с обычной очередью в магазине: новые посетители добавляются в конец очереди, а обслуживание начинается с того, кто первым стоит в начале.

Для нахождения всех возможных обходов графа с использованием очереди мы можем применить алгоритм поиска в ширину (BFS - Breadth-First Search). Алгоритм BFS обходит граф начиная с определенной вершины и посещает все вершины, смежные с данной. Таким образом алгоритм постепенно расширяет свой поиск и находит все возможные обходы.

# **Алгоритм метода:**

1. **Создайте очередь для хранения путей. Инициализируйте очередь путем, содержащим только стартовую вершину.**
2. Создайте пустой список для хранения всех обходов графа.
3. Установите начальное время выполнения алгоритма.
4. Пока очередь не пуста и время выполнения не превышает заданный лимит:
   * Извлеките из очереди путь.
   * Получите последнюю вершину этого пути.
   * Если длина пути равна количеству вершин в графе, добавьте этот путь в список обходов.
   * В противном случае, для каждого соседа последней вершины добавьте новый путь в очередь, состоящий из текущего пути плюс этот сосед.
5. Переберите все обходы и удалите из них повторяющиеся вершины.
6. Верните список всех обходов.

# **Описание программы:**

Программа реализована на языке Python 3.12 с использованием следующих пакетов: networkx, matplotlib.pyplot, streamlit, pandas, time. Полный список требований с казанными версиями можно найти в приложении. В программе используются функций. В таблицах описание функций программы.

*Таблица 1: функции*

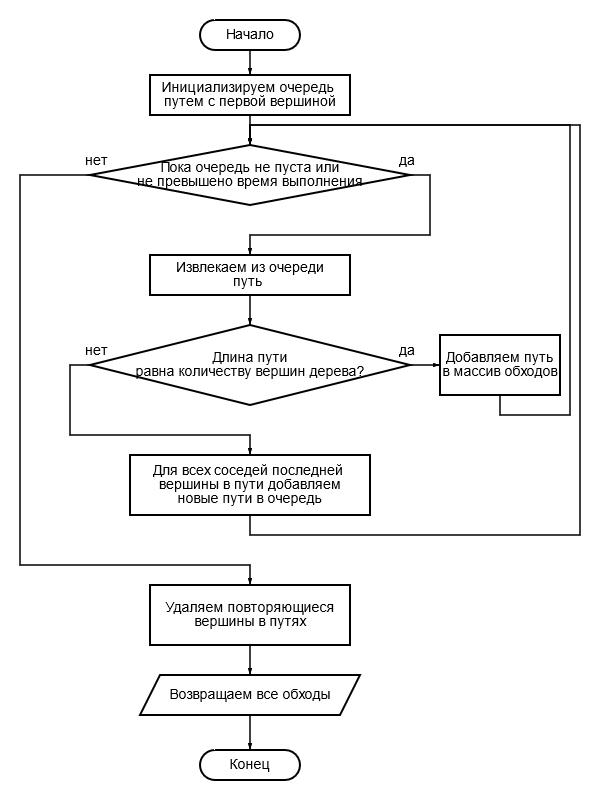
| Имя функции | Тип возвращаемого значения | Описание функции |
| --- | --- | --- |
| all\_traverses | list[list[int]] | Основная функция, которая принимает граф и начальную вершину и возвращает все возможные обходы графа из этой вершины |

*Таблица 2: Переменные*

| Имя переменной | Тип | Описание |
| --- | --- | --- |
| G | networkx.Graph | Граф, который создается пользователем в приложении Streamlit |
| start\_node | int | Начальная вершина для обхода в графе, выбираемая пользователем в приложении Streamlit |
| num\_nodes | int | Количество вершин в графе, введенное пользователем в приложении Streamlit |
| num\_edges | int | Количество ребер в графе, введенное пользователем в приложении Streamlit |
| edge | str | Строка, представляющая ребро графа, введенное пользователем в приложении Streamlit |
| node1, node2 | int | Вершины, связанные ребром, введенные пользователем в приложении Streamlit |
| graph | matplotlib.figure.Figure | Графическое представление графа, визуализируемого в приложении Streamlit |
| pos | dict | Словарь с позициями вершин графа для визуализации в приложении Streamlit |
| traverses | list[list[int]] | Список всех обходов графа, полученных с помощью функции all\_traverses |
| df | pandas.DataFrame | DataFrame, используемый для представления обходов графа в виде таблицы в приложении Streamlit |
| line | list[list[int]] | Очередь для хранения путей в графе |
| traverses | list[list[int]] | Список для хранения всех обходов графа |
| start\_time | float | Время начала выполнения алгоритма |
| path | list[int] | Путь в графе, извлеченный из очереди |
| node | int | Последняя вершина в текущем пути |
| neighbor | int | Соседняя вершина для текущей вершины в графе |
| ans | list[list[int]] | Список обходов, из которых удалены повторяющиеся вершины |
| traverse | list[int] | Один из обходов графа |
| tmp | list[int] | Временный список для хранения уникальных вершин в обходе |

# **Блок-схема:**

*Рис 1: блок-схема программы*

****

# **Рекомендации для пользователя:**

Запуск программы:

1. Убедитесь, что на вашем компьютере установлен Python.
2. Установите необходимые библиотеки: $ pip install -r requirements.txt
3. Запустите программу для активации графического интерфейса: $ streamlit run tree\_lab/all\_traverses\_display.py

Использование программы:

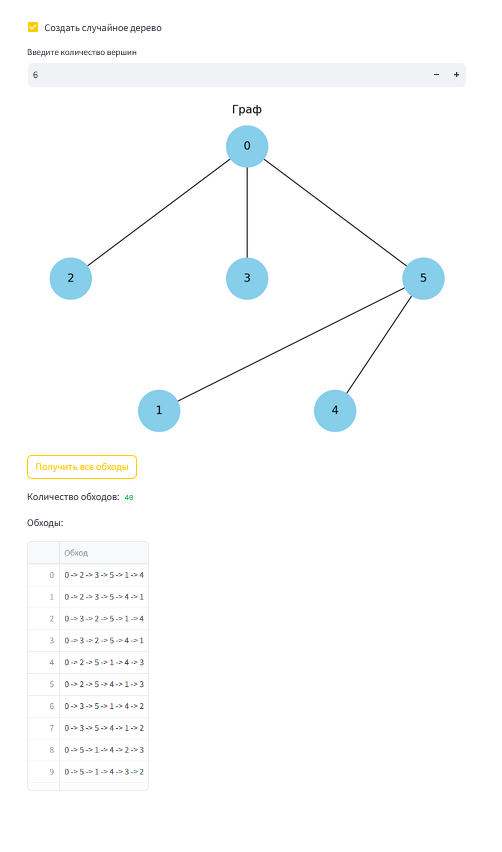
1. С помощью чекбокса выберите нужен ли случайный граф или вы хотите задать его сами.
2. Если вы задаете граф сами — введите количество вершин и ребра между вершинами (граф должен быть деревом).
3. Вычислите все возможные обходы дерева

Рекомендуется убедиться в правильности ввода значений параметров перед запуском алгоритма поиска всех возможных обходов дерева.

**Исходный код программы доступен по ссылке ниже:**

<https://github.com/StephanOlizko?tab=repositories>

# **Контрольный пример:**



*Рис 2: пример окна программы*

# **Вывод:**

В ходе выполнения лабораторной работы была рассмотрена задача о нахождении всех возможных обходов дерева и реализовано её решение с использованием структуры данных «очередь» и алгоритма BFS.

Таким образом, лабораторная работа позволила закрепить знания по работе с графами и деревьями, а также научиться применять алгоритмы поиска в ширину.

# **Источники:**

https://matplotlib.org/stable/index.html

<https://docs.streamlit.io/>

https://networkx.org/