Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

Выполнил студент группы КС-38 (Казанцев Леонард Антонович)

Ссылка на репозиторий: (https://github.com/SimpleMaking/Algorithms-6-sem-/tree/main/Lab\_2(сортировка\_part2))

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи: (10.03.2023)

Оглавление

[Описание задачи. 2](#_Toc127270410)

[Описание метода/модели. 3](#_Toc127270411)

[Выполнение задачи. 4](#_Toc127270412)

[Заключение. 4](#_Toc127270413)

# Описание задачи.

В рамках лабораторной работы необходимо реализовать генератор случайных графов, генератор должен содержать следующие параметры:

* Максимальное/Минимальное количество генерируемых вершин
* Максимальное/Минимальное количество генерируемых ребер
* Максимальное количество ребер связанных с одной вершины
* Генерируется ли направленный граф
* Максимальное количество входящих и выходящих ребер

Сгенерированный граф должен быть описан в рамках одного класса (этот класс не должен заниматься генерацией), и должен обладать обязательно следующими методами:

* Выдача матрицы смежности
* Выдача матрицы инцидентности
* Выдача список смежности
* Выдача списка ребер

В качестве проверки работоспособности, требуется сгенерировать 10 графов с возрастающим количеством вершин и ребер (количество выбирать в зависимости от сложности расчета для вашего отдельно взятого ПК). На каждом из сгенерированных графов требуется выполнить поиск кратчайшего пути или подтвердить его отсутствие из точки А в точку Б, выбирающиеся случайным образом заранее, поиском в ширину и поиском в глубину, замерев время требуемое на выполнение операции. Результаты замеров наложить на график и проанализировать эффективность применения обоих методов к этой задаче.

# Описание метода/модели.

Граф — математическая абстракция реальной системы любой природы, объекты которой обладают парными связями. Граф как математический объект есть совокупность двух множеств — множества самих объектов, называемого множеством вершин, и множества их парных связей, называемого множеством рёбер.

Метод поиска в глубину

**Поиск в глубину** (англ. *Depth-first search,****DFS***) — один из методов обхода графа. Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра.

Метод поиска в ширину

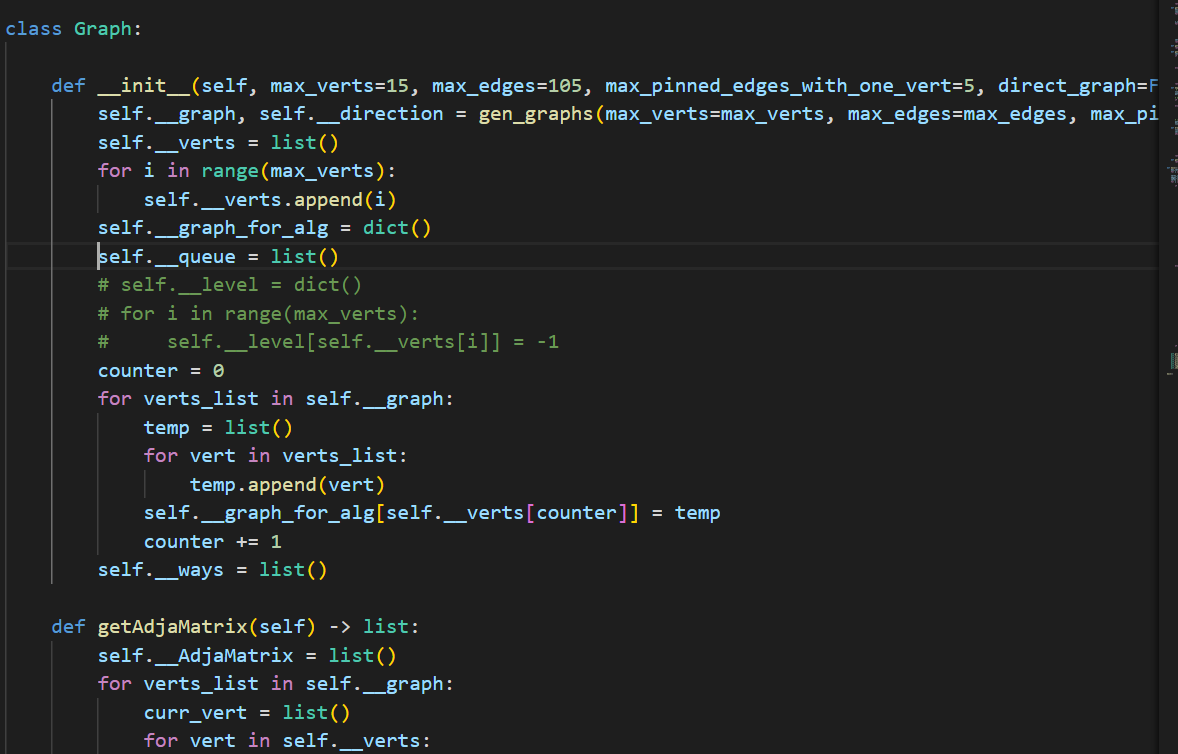
**Поиск в ширину** (англ. *breadth-first search*, **BFS**) — один из методов обхода графа. Пусть задан граф �=(�,�) и выделена исходная вершина �. Алгоритм поиска в ширину систематически обходит все ребра � для «открытия» всех вершин, достижимых из �, вычисляя при этом расстояние (минимальное количество рёбер) от � до каждой достижимой из � вершины. Алгоритм работает как для ориентированных, так и для неориентированных графов.

# Выполнение задачи.

Для решения поставленной задачи был выбран язык программирования python. Граф реализован в виде класса с методами получения матрицы смежности, матрицы инцидентности, списка смежности и списка ребер.

Код класса:

* 1. В конструкторе класса получаем сгенерированный граф.
  2. Создаем массив вершин.
  3. Создаем представление графа в виде словаре, где ключ - это вершина, а значение это список смежных вершин.
  4. Функция получения матрицы смежности.



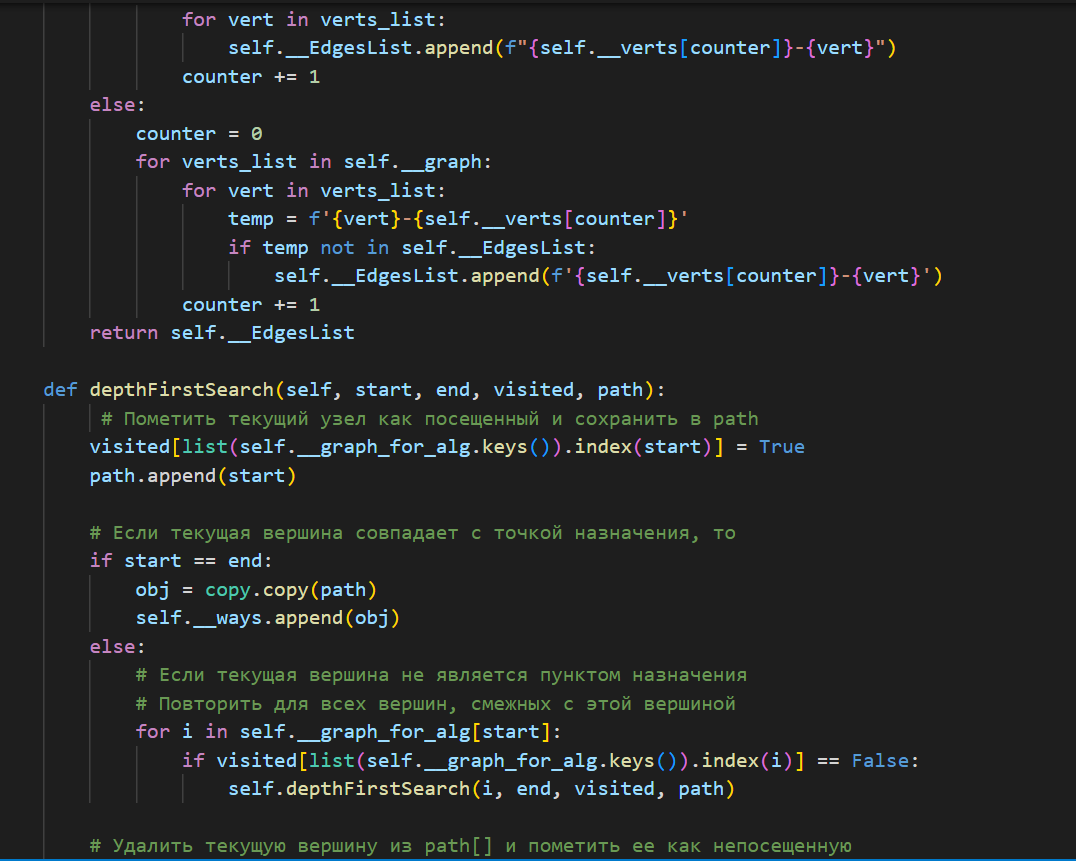
* 1. Функция получения матрицы инцидентности.
  2. Получение списка смежности (возвращается функцией генерации графа).



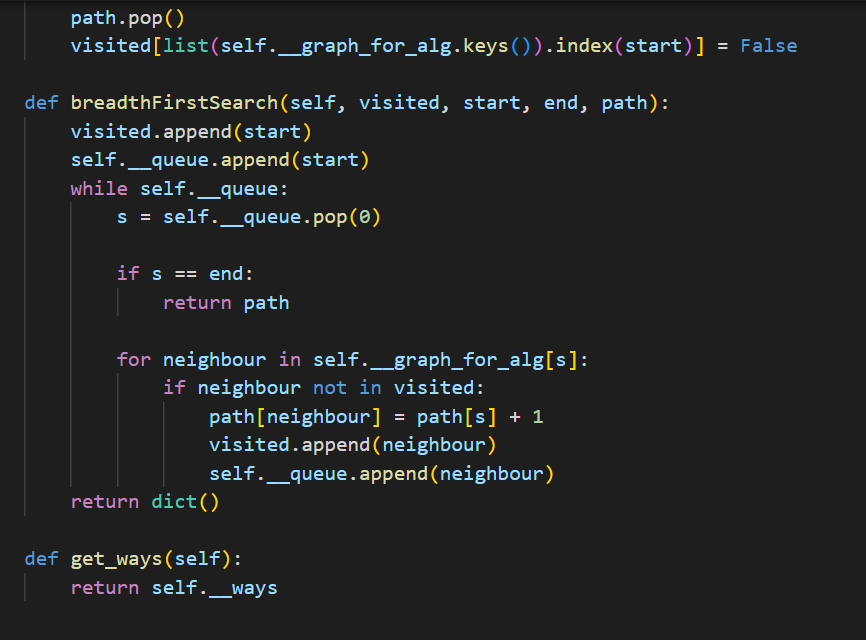
* 1. Получение списка ребер (для направленного и ненаправленного графа).

Различие в том, что у ненаправленного графа не может быть два разных противоположных ребра, такие ребра считаются за одно.

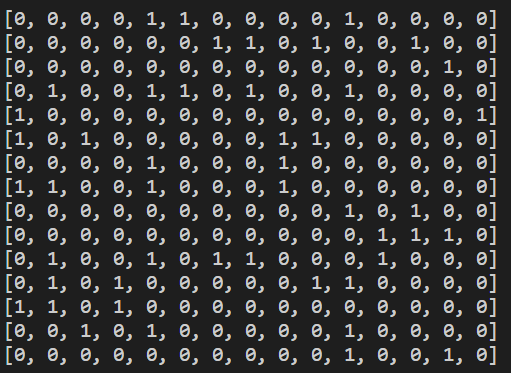
* 1. Рекурсивный алгоритм поиска в глубину. Он просматривает все возможные пути в графе.



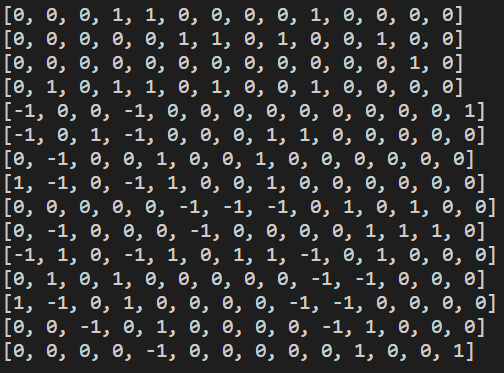
* 1. Алгоритм поиска в ширину. Он просматривает всех соседей от исходной вершины и, если в процессе этого действия текущая вершина окажется равной конечной, то алгоритм прекращает работу. Главная сущность этого алгоритма – очередь вершин, которые еще не были посещены в процессе алгоритма.



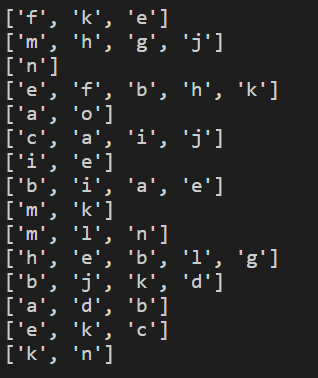
Генерация графов производилась в отдельном файле функцией gen\_graphs(). Рассмотрим функционал класса и работу функции генерации: Для теста взял направленный граф из 15 вершин, каждая из них представлена буквами латинского алфавита от a до o. Вывод всех четырех представлений данного графа:



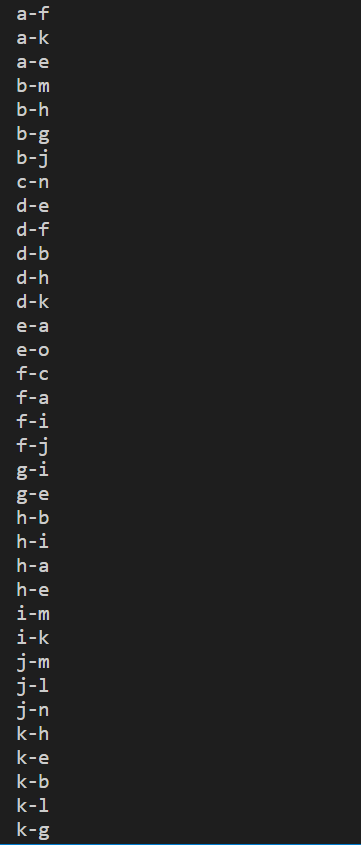
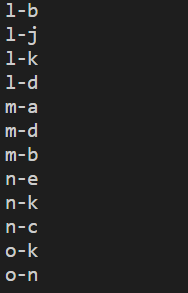
Матрица смежности



Матрица инцидентности

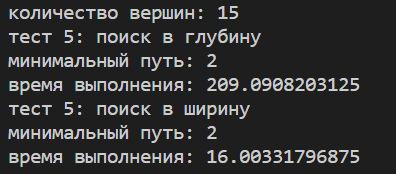


Список смежности

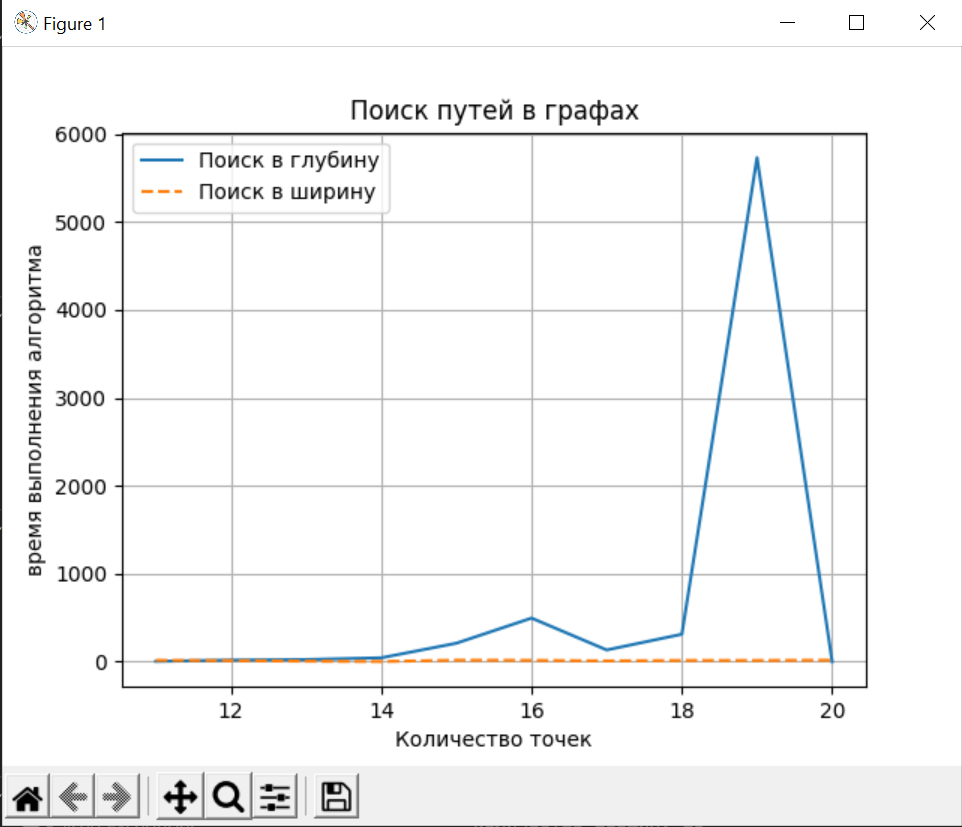
 

Список ребер

Далее генерируются 10 графов с разным количеством вершин и ребер. У каждого из них с помощью двух алгоритмов находится длина кратчайшего пути от начальной точки до конечной. Время в миллисекундах. Приведу пример вывода:



Поиск кратчайшего пути между случайными точками для графа с 15 вершинами



Пути в 10 графах

# Заключение.

Реализация графов и их генерация оказалась не очень сложной. Видно, что метод поиска в глубину работает дольше, чем метод поиска в ширину. Поиск в глубину задействует рекурсию и просматривает все возможные пути от каждой точки до каждой, а метод поиска в ширину обходит граф внешним образом, используя очередь вершин и список посещенных вершин. Задание выполнено полностью.