**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-36 Перминова П.А.

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи:

# Оглавление

[**Оглавление**](#_r3cf0m8tbo4x) **2**

[**Описание задачи**](#_m4x1kvycbh5w) **3**

# 

# Описание задачи

Необходимо реализовать генератор случайных графов, генератор должен содержать следующие параметры:

⦁ Максимальное/Минимальное количество генерируемых вершин

⦁ Максимальное/Минимальное количество генерируемых ребер

⦁ Максимальное количество рёбер, 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

Обход графа — это переход от одной его вершины к другой в поисках свойств связей этих вершин. Связи (линии, соединяющие вершины) называются направлениями, путями, гранями или ребрами графа. Вершины графа также именуются узлами.

Двумя основными алгоритмами обхода графа являются поиск в глубину (Depth-First Search, DFS) и поиск в ширину (Breadth-First Search, BFS). Ими мы и воспользуемся в данной лабораторной работе.

Несмотря на то, что оба алгоритма используются для обхода графа, они имеют некоторые отличия.

**Поиск в глубину**

DFS следует концепции «погружайся глубже, головой вперед» («go deep, head first»). Идея заключается в том, что мы двигаемся от начальной вершины (точки, места) в определенном направлении (по определенному пути) до тех пор, пока не достигнем конца пути или пункта назначения (искомой вершины). Если мы достигли конца пути, но он не является пунктом назначения, то мы возвращаемся назад (к точке разветвления или расхождения путей) и идем по другому маршруту.

**Поиск в ширину**

BFS (Breadth first search) следует концепции «расширяйся, поднимаясь на высоту птичьего полета» («go wide, bird’s eye-view»). Вместо того, чтобы двигаться по определенному пути до конца, BFS предполагает движение вперед по одному соседу за раз. Вместо следования по пути, BFS подразумевает посещение ближайших к s соседей за одно действие (шаг), затем посещение соседей соседей и так до тех пор, пока не будет обнаружено t.

Алгоритм DFS:

⦁ Выбираем любую вершину из еще не пройденных, обозначим ее как u.

⦁ Запускаем процедуру dfs(u)

⦁ Помечаем вершину u как пройденную

⦁ Для каждой не пройденной смежной с u вершиной (назовем ее v) запускаем dfs(v)

⦁ Повторяем шаги 1 и 2, пока все вершины не окажутся пройденными.

Сложность алгоритма определяется количеством вершин и количеством ребер в графе, вся процедура вызывается для каждой вершины не более одного раза, а в рамках работы процедуры рассматриваются все ребра, исходящие из вершины.

Алгоритм BFS:

⦁ Поместить узел, с которого начинается поиск, в изначально пустую очередь.

⦁ Извлечь из начала очереди узел u и пометить его как развёрнутый.

⦁ Если узел u является целевым узлом, то завершить поиск с результатом «успех».

⦁ В противном случае, в конец очереди добавляются все преемники узла u, которые ещё не развёрнуты и не находятся в очереди.

⦁ Если очередь пуста, то все узлы связного графа были просмотрены, следовательно, целевой узел недостижим из начального; завершить поиск с результатом «неудача».

⦁ Вернуться к п. 2.

Сложность алгоритма определяется количеством вершин и количеством ребер в графе, вся процедура вызывается для каждой вершины не более одного раза, а в рамках работы процедуры рассматриваются все ребра, исходящие из вершины.

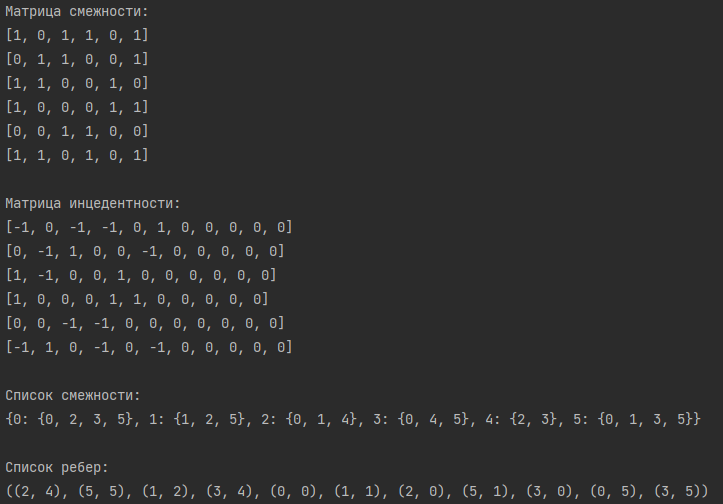
# Выполнение задачи

Данная лабораторная выполнена на языке Python.

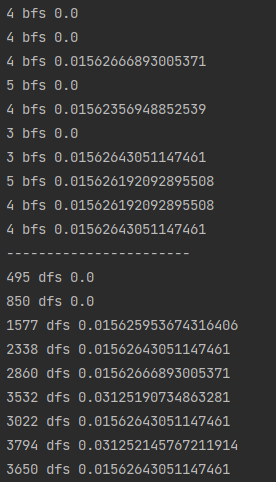
Классы, реализованные в программе:

* Generate (при его объявлении находящийся в нем метод generate\_graph() генерирует новый граф с заданными условиями)
* Graph (из реализованных в данном классе методов get\_adjacency\_matrix(), get\_incidence\_matrix(), get\_adjacency\_list() и get\_ribs\_list() можно получить разные виды представления графа: матрица смежности, матрица инцидентности, список смежности и список ребер)

Далее представлен вывод программы, на котором представлены все виды представления графа.

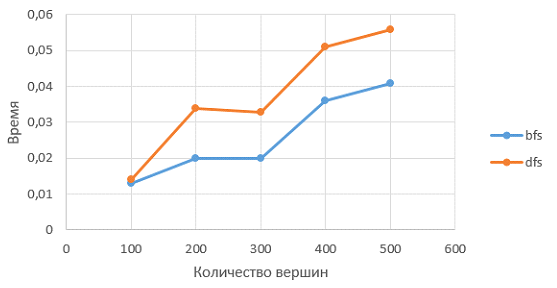


В функции analys() происходит генерация 10 разных графов, при этом каждый из последующих больше предыдущего. Генерируются точки A и B для обхода по графу. После этого в случайном порядке происходит обход по графу методом DFS или BFS. Результаты работы алгоритмов представлены ниже.



Можно заметить, что длина пути для BFS не сильно различается, в то время как для DFS длина пути увеличивается с увеличением количества вершин.

Результаты времени работы алгоритмов представлены в виде графика.



Если сравнить данный графики, то можно увидеть, что графики идут почти параллельно, однако для DFS понадобиться большее количество времени, чем для BFS. Так же, вне зависимости от метода, время работы алгоритмов растет с ростом количества вершин.

# Заключение

В ходе выполнения лабораторной работы был сделан следующий вывод: BFS является более стабильным, так как в BFS переполнение очереди почти невозможно, а в DFS при большом количестве ребер и вершин происходит переполнение стека. Для проверки существования пути лучше подойдет DFS, но, так как DFS не всегда может выдать точный результат, для поиска оптимального пути лучше подойдет BFS.

Однако DFS может выдать результат лучше, если вторая искомая вершина будет, например, на конце иерархического дерева. Также можно увидеть следующую закономерность: при получении хотя бы одного представления графа можно выразить все остальные, но для этого необходимо жестко регулировать входные данные, так как, например, для матрицы инцидентности особенно важен порядок вершин в пути.