**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-30 Лобачев Дмитрий Сергеевич

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/DSLobachev\_30/blob/main/Algorithms/Laba4/Laba4.cpp

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 24.03.2023

**Оглавление**

[Описание задачи 3](#_Toc130500449)

[Описание метода/модели 4](#_Toc130500450)

[*Поиск в глубину* 4](#_Toc130500451)

[*Поиск в ширину* 5](#_Toc130500452)

[Выполнение задачи. 6](#_Toc130500453)

[Заключение 15](#_Toc130500454)

# 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.

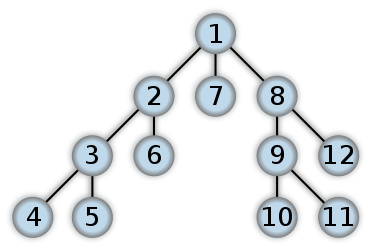
# Описание метода/модели

Граф — это абстрактный тип данных, который содержит вершины и грани. Выделяются такие типы графов, как:

* Ориентированный граф – граф, рёбрам которого присвоено направление;
* Неориентированный граф – граф, рёбра которого не имеют направления

## *Поиск в глубину*

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. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра.



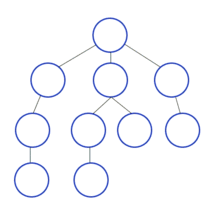
Пример алгоритма, работающий через стек:

1. Берем любую вершину графа и помещаем на вершину стека
2. Вытаскиваем из стека вершину и отмечаем её как посещенную
3. Создаем список смежных вершин для данной вершины. Вершины, которые мы не посещали, помещаем на вершину стека
4. Повторяем 2-3, пока наш стек не опустеет

Таким образом, мы получим список тех вершин, в которые мы можем попасть из начальной вершины. Отсюда мы можем сделать вывод, существует ли путь из вершины А в вершину Б (где А и Б заранее случайно выбранные вершины).

## *Поиск в ширину*

Поиск в ширину работает путём последовательного просмотра отдельных уровней графа.



Пример алгоритма, работающий через очередь:

1. Берем любую вершину графа и помещаем в очередь
2. Вытаскиваем из очереди вершину и отмечаем её как посещенную
3. Проверяем, есть ли смежные с ней вершины, которые мы еще не обнаружили. Если есть, то отмечаем ее как обнаруженную и добавляем в очередь
4. Повторяем 2-3, пока очередь не опустеет

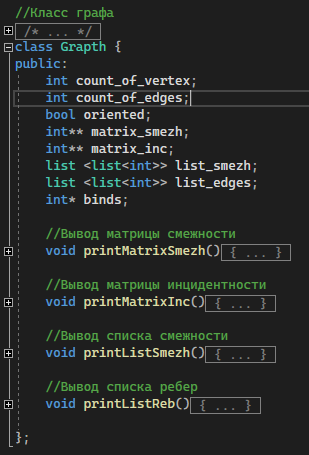
Таким образом, мы получили максимально короткий путь от вершины А в вершину Б (де А и Б заранее случайно выбранные вершины).

# Выполнение задачи.

Для реализации данного метода сортировки использовался язык программирования C++.

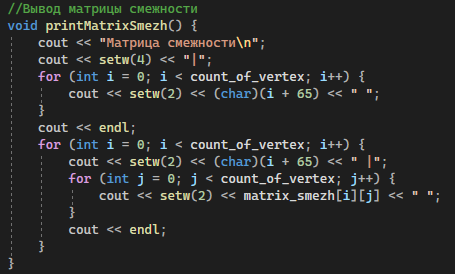
**Граф**

1. Класс стека имеет поля: количество вершин(count\_of\_vertex), количество ребер(count\_of\_edges), ориентированность графа(oriented), матрица смежности(matrix smezh), матрица инцидентности(matrix\_inc), список смежности(list\_smezh), список ребер(list\_edges) и количество связей(binds).

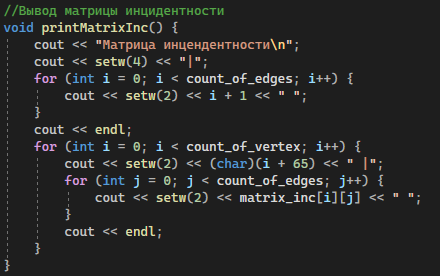


1. Класс графа имеет несколько необходимых нам функций:

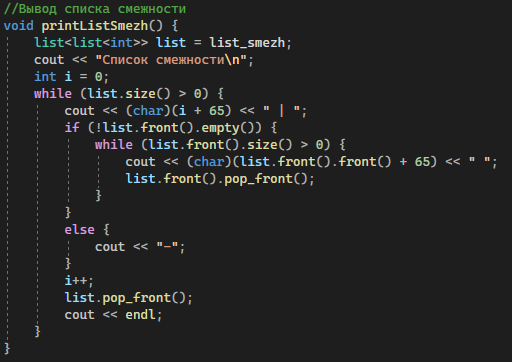
* Функция вывода матрицы смежности printMatrixSmezh()



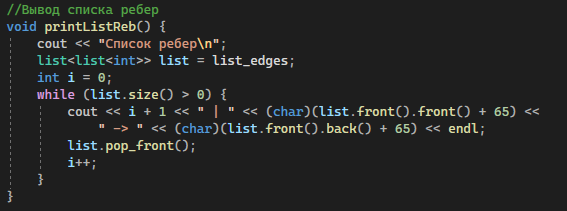
* Функция вывода матрицы инцидентности printMatrixInc()



* Функция вывода списка смежности printListSmezh()

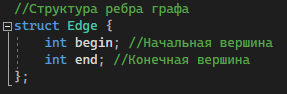


* Функция вывода списка ребер printListReb()



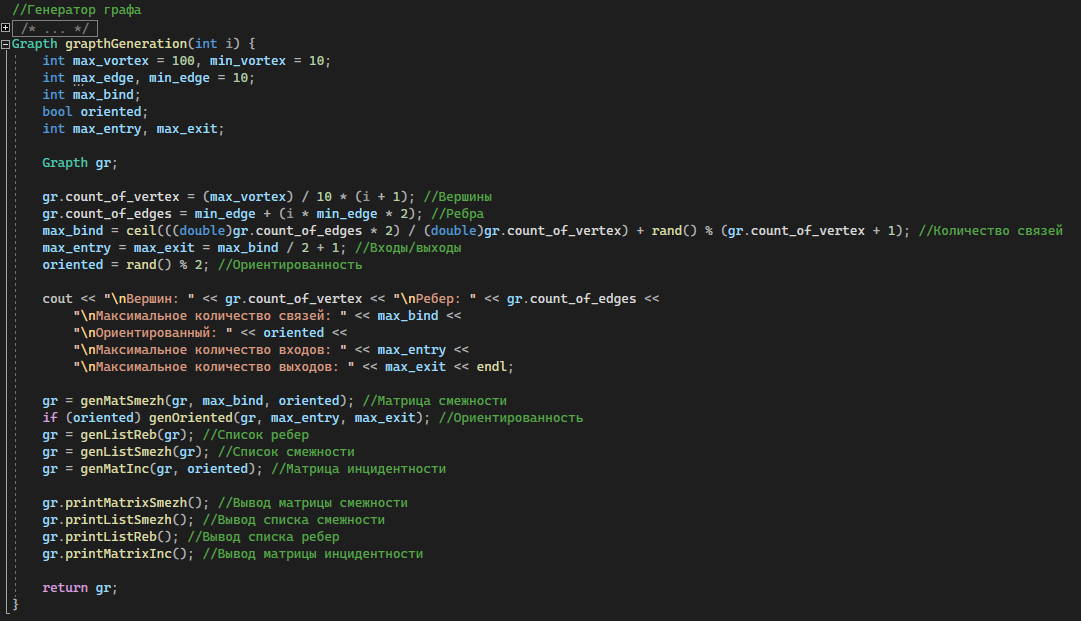
**Структура ребра**

Была написана структура ребра, хранящая в себе начальную (begin) и конечную(end) вершины

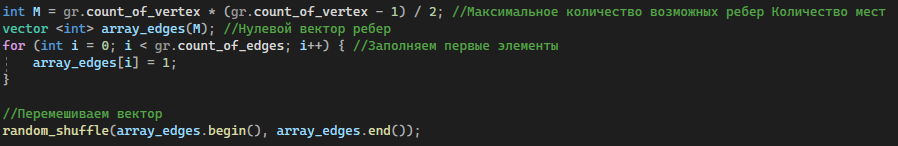


**Генерация графа**

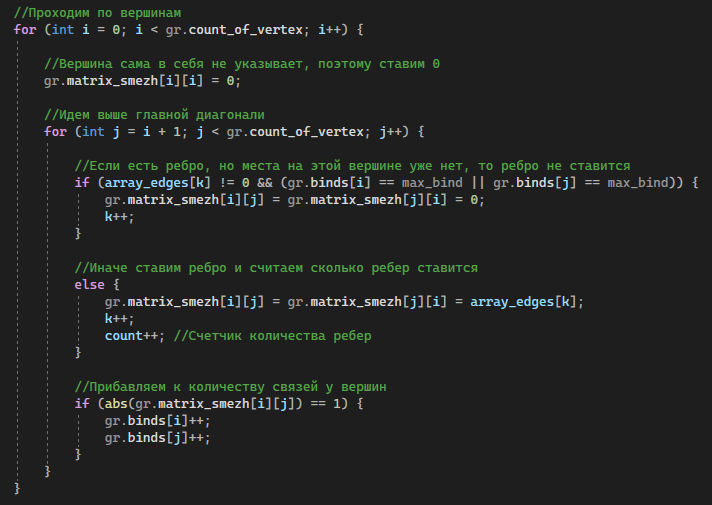
1. Был реализован генератор графов grapthGeneration(), в котором генерируются количество вершин, ребер, максимальное количество связей, максимальное количество входящих и выходящих ребер и ориентированность графа.



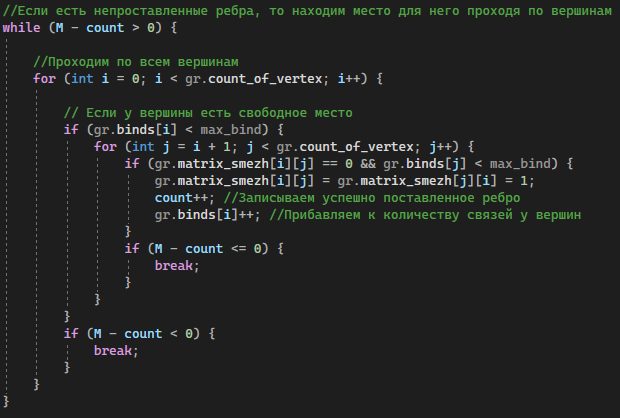
1. Функция genMatSmezh() генерирует матрицу смежности для определенного графа. Сперва генерируется вектор из M элементов (где М – максимальное количество ребер), первые N элементов которого равны 1 (где N – количество ребер в графе), а остальные равны 0. Далее этот вектор перемешивается для случайной расстановки функцией random\_shuffle



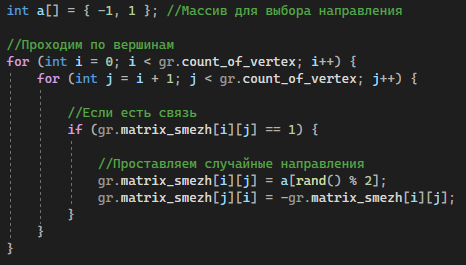
Следующим шагом заполняется матрица смежности так, что элементы вектора располагались выше главной диагонали (в исследуемых графах не существует ребер, входящие в исходящую вершину). Ниже главной диагонали зеркально отображаются эти же элементы. Параллельно этому процессу идет проверка на максимальное количество связей у вершины.



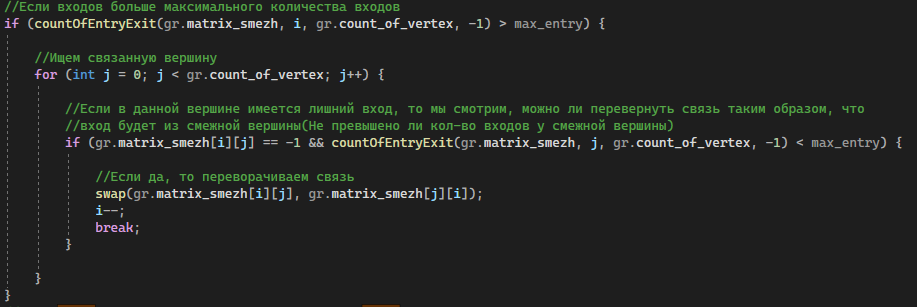
После расстановки всех ребер, выставляются те ребра, которые оказались лишними из за превышения максимального количества ребер у одной вершины



С помощью функции genOriented() мы расставляем направления в графе, если он оказался ориентированным (1 – выход, -1 – вход).

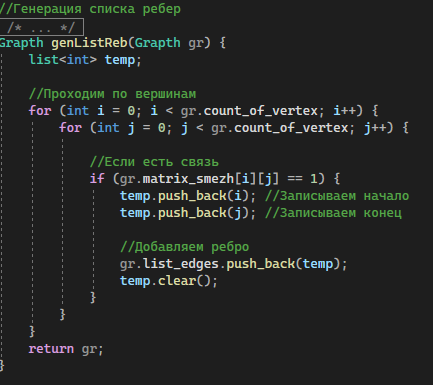


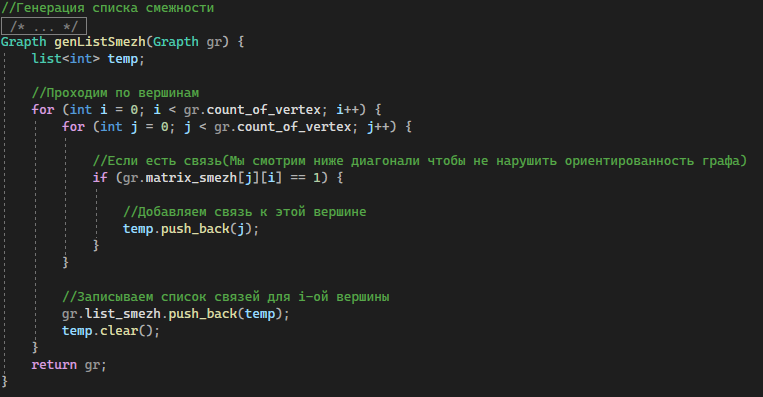
Далее выполняется проверка на максимальное количество входов и выходов у данной вершины и если необходимо, меняем направление



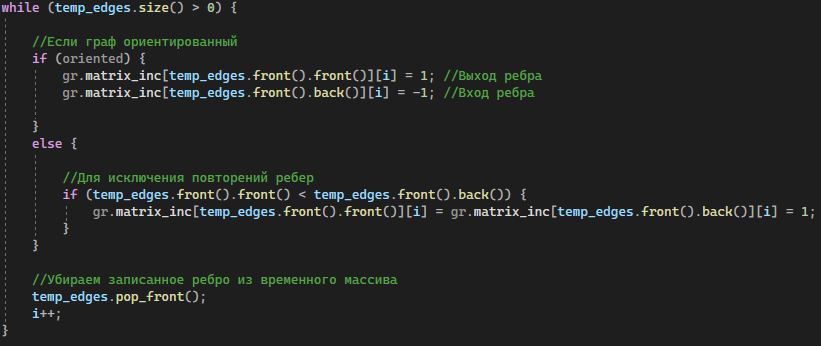
После этого заменяем все -1 на 0 для правильного отображения.

1. Функция genListReb() генерирует список ребер на основе матрицы смежности. В буферный список temp записываются начальная(i) и конечная(j) вершины ребра.



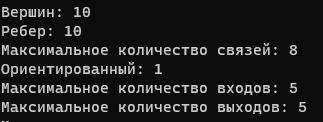
1. Функция genListSmezh() генерирует список смежности на основе матрицы смежности. В буферный спискок temp записываются вершины, которые смежны с данной вершиной
2. Функция genMatInc() генерирует матрицу инцидентности на основе списка ребер.

Берется верхний список вершин из списка ребер. После заполняется начальная и конечная вершины у ребра. Для ориентированного графа 1 и -1, для неориентированного 1 и 1. Таким образом данная операция выполняется, пока цикл не дойдет до конца списка.



**Выполнение работы**

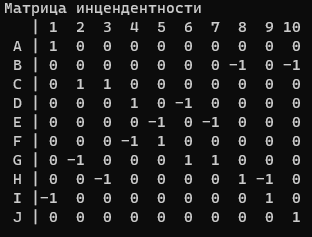
Была сгенерирован граф и выведена информация в консоль



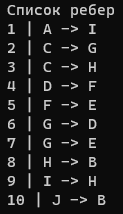
* Матрица смежности



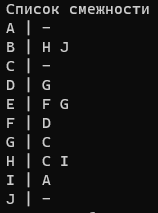
* Матрица инцидентности



* Список ребер

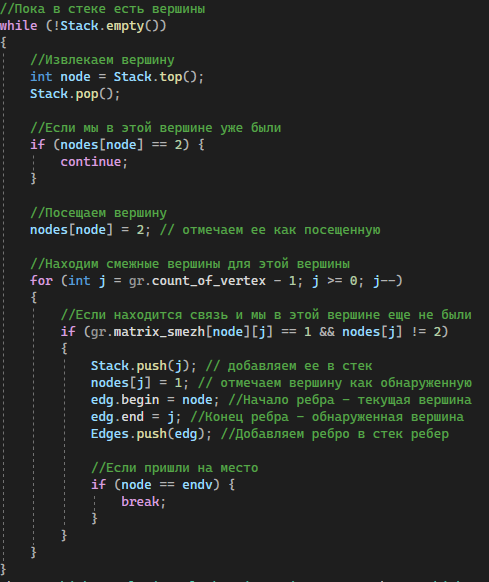


* Список смежности

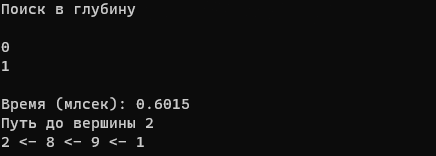


Далее мы приступаем к поиску в глубину из вершину А в вершину Б, где А и Б – случайные вершины.

1. Создается массив размером равным количеству вершин. Изначально все элементы равны 0. Данный массив будет показывать нам посетили ли мы ту или иную вершину или нет.
2. Генерируем начальную и конечную вершины.
3. Помещаем начальную вершину в стек.
4. Извлекаем верхний элемент из стека и отмечаем его как посещенный.
5. Проверяем для взятой вершины все смежные и не обнаруженные вершины. Если такие имеются, то добавляем ее в стек и помечаем как обнаруженную.
6. Повторяем с 4 пункта, пока стек не будет пустым.

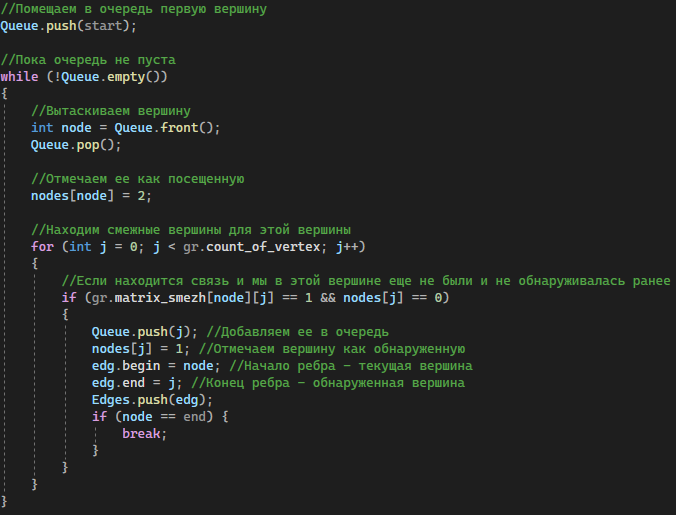


Идем из вершины 1 в вершину 2

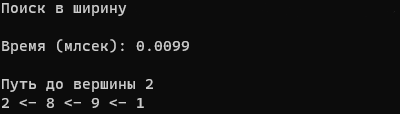


После этого приступаем к поиску в ширину, для определения кратчайшего пути.

1. Создается массив размером равным количеству вершин. Изначально все элементы равны 0. Данный массив будет показывать нам посетили ли мы ту или иную вершину или нет.
2. Помещаем начальную вершину в очередь.
3. Извлекаем элемент из очереди и отмечаем его как посещенный.
4. Проверяем для взятой вершины все смежные и не обнаруженные вершины. Если такие имеются, то добавляем ее в стек и помечаем как обнаруженную.
5. Повторяем с 4 пункта, пока стек не будет пустым.



Идем из вершины 1 в вершину 2



Для каждого поиска было замерено время работы в миллисекундах. Полученные результаты были зафиксированы и на их основе были построены графики зависимостей времени выполнения алгоритма от количества ребер и вершин.



# Заключение

Проанализировав графики, можно заметит, что скорость нахождения пути практически не зависит от количества вершин и ребер у графа. Также можно отметить, что скорость нахождения пути с помощью поиска в ширину в несколько десятков раз меньше, чем скорость нахождения пути методом поиска в глубину. Также заметно, что ориентированность графа также не влияет на скорость нахождения пути.