**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

# **Дисциплина:** Алгоритмы и структуры данных

Отчет

Лабораторная работа 7. Графы

Выполнил:

**Сулейманов Руслан**

Группа:

**K33402**

Проверил:

Иванов С. Е.

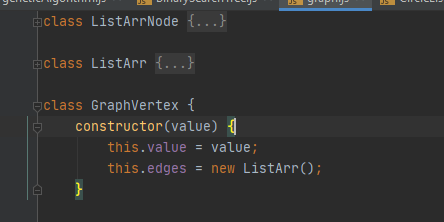
Санкт-Петербург

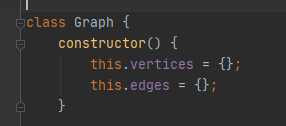
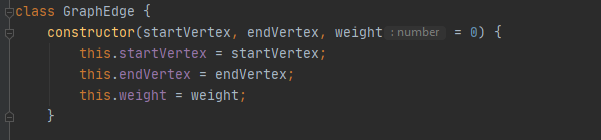
2022 г.

**Цель работы**: Изучить алгоритмы на графах..

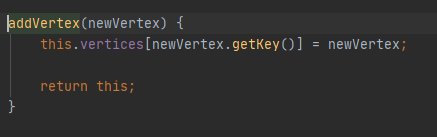
**Ход работы**:

Первое что нужно сделать, продумать архитектуру графа.   
Будем хранить ребра и вершины графа. Создадим класс для ребра, который будет включать в себя 2 вершины и вес ребра, и класс для вершины, в котором будем хранить значение вершины и его соседей, для удобства вторую сущность будем хранить в классе связного списка, который реализовывали во 2 лабораторной работе.



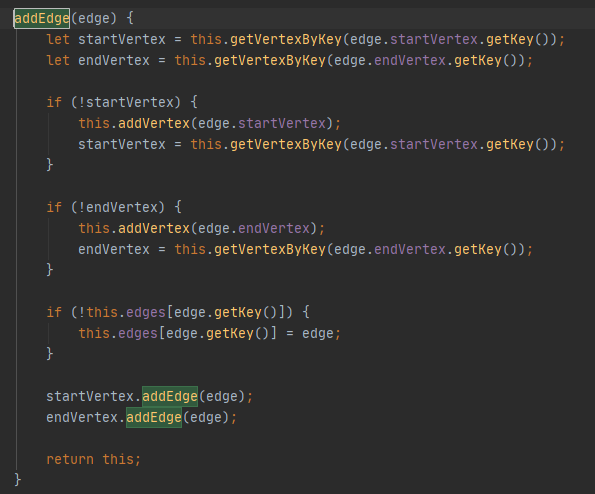


Начнем с добавления вершин в граф:



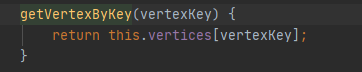
Так как мы добавляем элемент класса вершины, мы используем метод класса для получения ключа, и по этому ключу добавляем вершину.

Аналогично, создадим элементы класса Edge, и добавим их к классу Graph



Первое что мы делаем ищем есть ли вершины, если нет, то создаем их.

Первые две строчки кода могут показаться странными, зачем были использован такой сложный путь, причина в том, что метод getVertexByKey -написан в рамках графа, и вернет значения графа, а edge.startVertex – может существовать и не в класса графа.

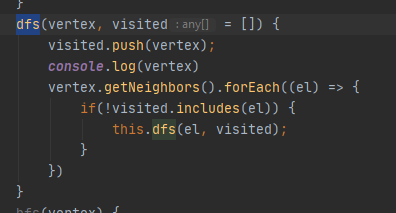


Раз мы научились создавать вершины и ребра, создадим класс и реализуем картинку из задания:



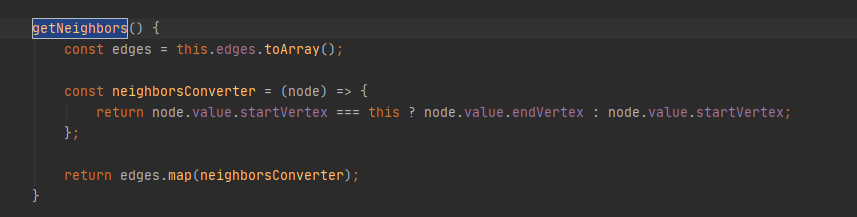


Базовый шаблон класса описан.   
  
1) Поиск в глубину и ширину

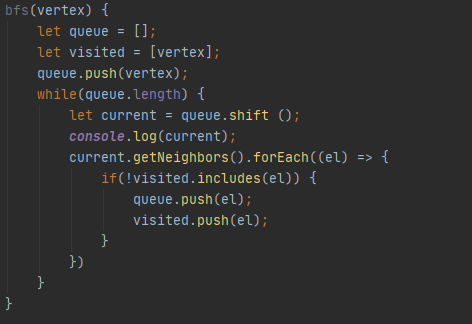


Данный метод на вход получает вершину, после чего получает всех соседей данной вершины и для каждого соседа вызывает рекурсивно эту функцию, пока все элементы не будут посещены.

Метод поиска соседей:



Поиск в ширину – отличается тем, что сперва пройдет по всем соседям, и только потом пойдет дальше:

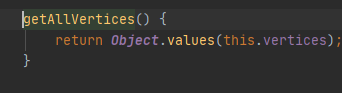
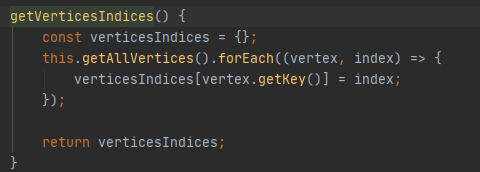


Для этого сделаем аналогично прошлой задачи массив посещенных вершин, а также очередь, куда будем добавлять и удалять посещенную ноду, тем самым с каждым ходом мы будем пополнять очередь на всех соседей, которые еще не были посещены.

2)



Для реализации кротчайших путей было реализовано два дополнительных метода, для создании матрицы и получение всех индексов: 

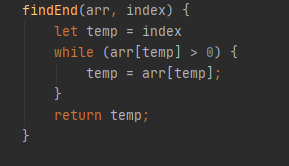


Мы заполняем матрицы бесконечными значениями, а затем на пересечении узлов ставим их вес. Таким образом у нас есть матрица графа: создадим еще один массив с длинами до каждой точки:

Поместим в стартовую позицию нулевое значение и пропишем сумму длин до всех вершин. Таким образом, у нас будет массив кратчайших путей до всех вершин, если подать конечную точку, то можно узнать кратчайшее расстояние до нее.

3)

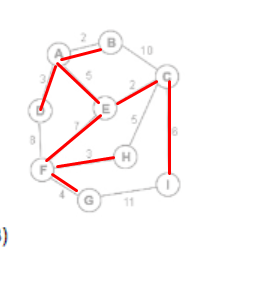
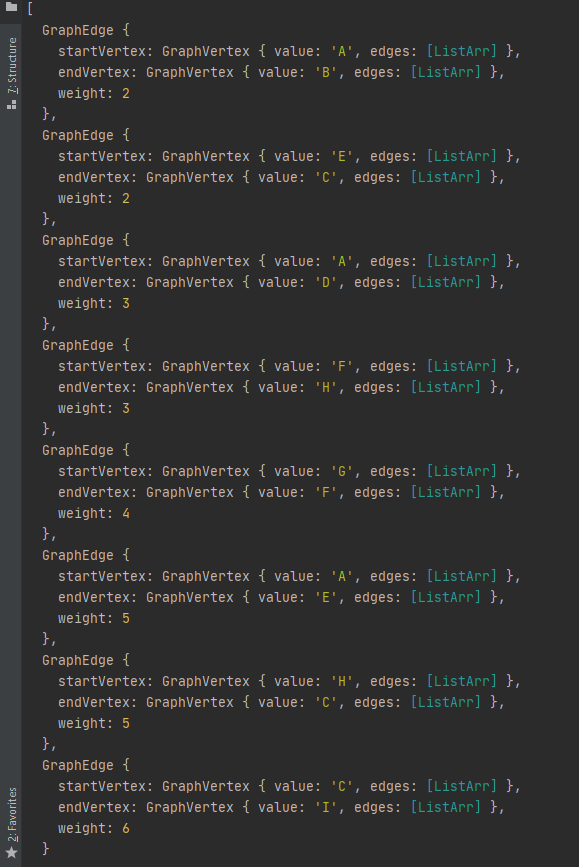




Для метода Краскала, получим все ребра, отсортируем их в порядке возрастания веса, создадим пустые массивы, для ответа и для выбранных значений, пройдемся массивом по всем ребрам проверим что данное ребро не замыкает дерево. И если все хорошо, то добавим ребро в ответ и в укороченный массив ответов.

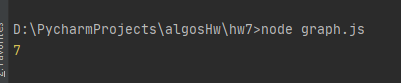
Результаты:

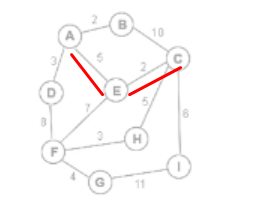
3)



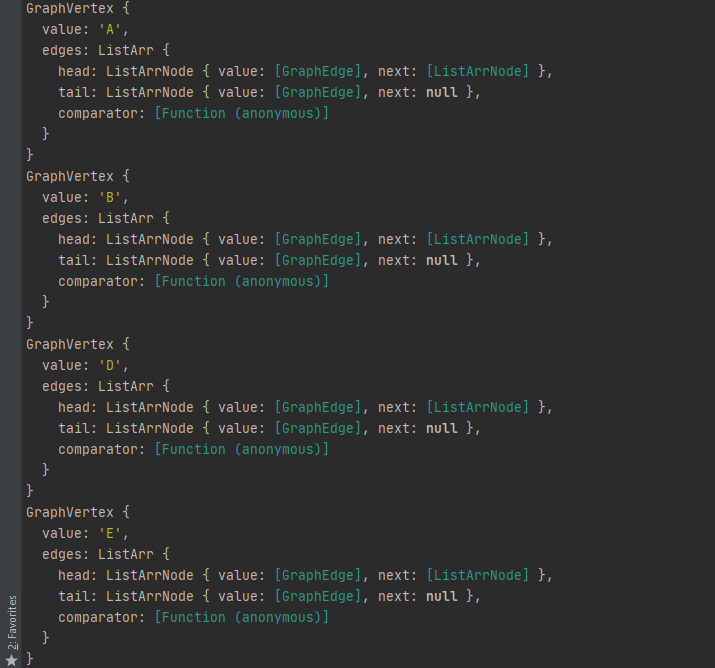
**2)**







1. Поиск в ширину от точки А



Поиск в длину от точки A:



**Вывод**: Изучены новые алгоритмы для работы с графами, написаны классы, возможно, для будущих работ, как это было сделано со связным списком.