**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЁТ**

**по лабораторной работе №2**

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

**Тема: Алгоритм Дейкстры**

Студент гр. 0322 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шаронин А.Д.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пестерев Д.О.

Санкт-Петербург

2023

# Задача

Реализовать алгоритм Дейкстры с помощью выбранной базовой структуры данных, двоичной кучи и Фибоначчиевой кучи (кучи реализовать самостоятельно). Сравнить временные затраты на выполнение данных реализаций алгоритма для графов низкой/средней/высокой связности. Отчет должен содержать графики и выводы по проделанной работе.

# Описание реализуемых классов и структур

binaryheap — структура двоичной кучи, узлы соединены в виде дерева, где каждого родителя не более двух потомков, реализовано в виде вектора, где родитель n, а его потомки 2\*n+1 и 2\*n+2.

node — структура узла фибоначчиевой кучи.

FibonacciHeap — класс фибоначчиевой кучи.

**Сравнение временных затрат**

На вход подаётся количество вершин в графе, вершина, из которой хотим найти пути до всех остальных вершин, а также расстояния от каждой вершины до каждой вершины. Для небольших графов не так страшно, но граф из 10 вершин уже потребует 100 значений длин путей. Поэтому можно просто скопировать либо длинную строчку, где все длины разделены пробелом, либо матрицу n\*n, где n количество вершин (нагляднее на рисунках ниже), ввод спокойно это примет и правильно всё запишет. Для сравнения будем использовать граф из 100 вершин, а искать путь 42 вершины (взято наугад, почти). Также была реализована ещё одна программа, которая выдаёт матрицу n\*n, а все значения по главной диагонали равны 0, это показывает, что путь из вершины в саму себя равен 0, также в этой программе можно примерно менять количество нулей в матрице, которые показывают, что пути из одной вершины в другую нет, соответственно, чем меньше нулей, тем связнее граф и наоборот.

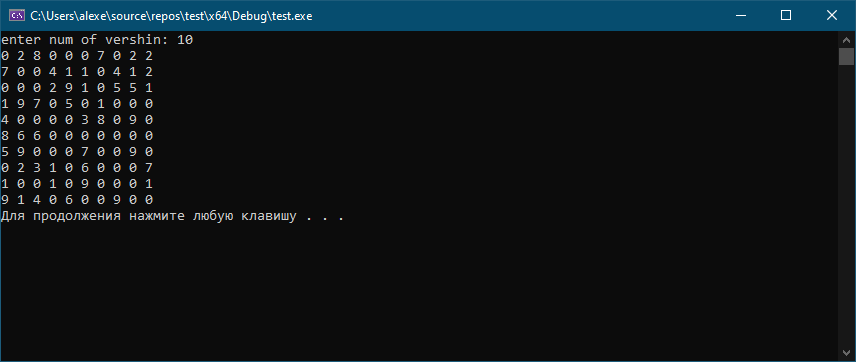
****

Рисунок 1 — граф из 10 вершин, с примерно 50% нулями в нём

Был взят граф из 100 вершин, поиск осуществлялся для 42 вершины. Были взяты значения для 11 случаев связности графов, для 0, 10, 20, 30, 40, 50, 60, 70, 80, 80, 100% нулей (отсутствие связи между вершинами). Каждый алгоритм (их два, по одному на кучу) был вызван 1500 раз и взято среднее значение времени работы алгоритма.

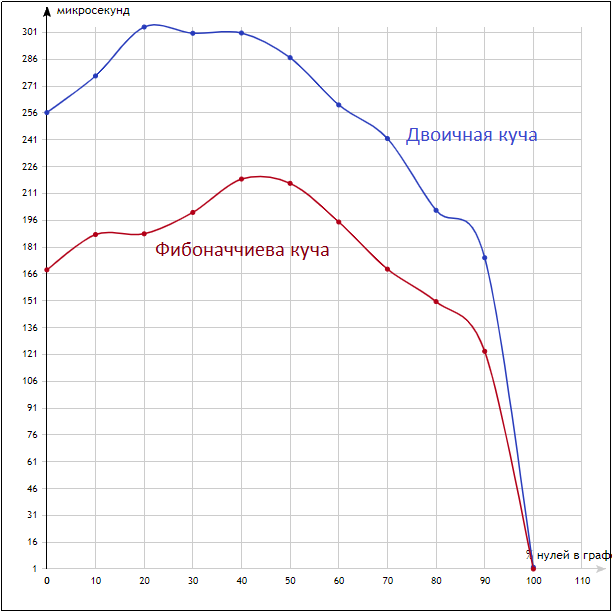


Рисунок 2 — зависимость времени работы алгоритма(в мкс) от связности графа

Как видно из графика алгоритм Дейкстры с Фибоначчиевой кучей тратит куда меньше времени для поиска пути, нежели алгоритм с двоичной кучей. Оно и не удивительно, а алгоритме каждая куча одинаковое количество раз вставляет элемент в кучу, находит минимум, удаляет минимум и проверяет кучу на пустоту, значит время работы будет зависеть от того, сколько времени занимают функции вставки, удаления минимума, нахождения минимума и проверки на пустоту. Для двоичной кучи вставка и удаление происходят за O(log(n)), где n — количество элементов вводимой матрицы (в нашем случае это количество вершин графа в квадрате), а поиск минимума и проверка на пустоту за O(1). Для Фибоначчиевой кучи всё происходит за O(1), кроме удаления минимума, оно за O(log(n)). Казалось бы, отличие только во вставке, но вставлять нужно огромное количество значений (для нашего графа из 100 вершин — это почти 10000 значений), вот тут двоичная куча и проигрывает. Сами алгоритмы Дейкстры для обеих куч идентичны в своей реализации.

**Пример работы программы**

Возьмём к примеру этот граф, который я отыскал на просторах гугла. Он неориентированный.

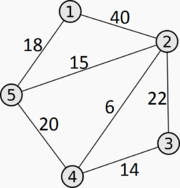


Рисунок 3 — неориентированный граф из 5 вершин

И возьмём 2-ую вершину как стартовую. Расстояния до других вершин будут равны: 2 -> 1 = 33, 2 -> 2 = 0, 2 -> 3 = 20, 2 -> 4 = 6, 2 -> 5 = 15. Запустим программу и введём значения:

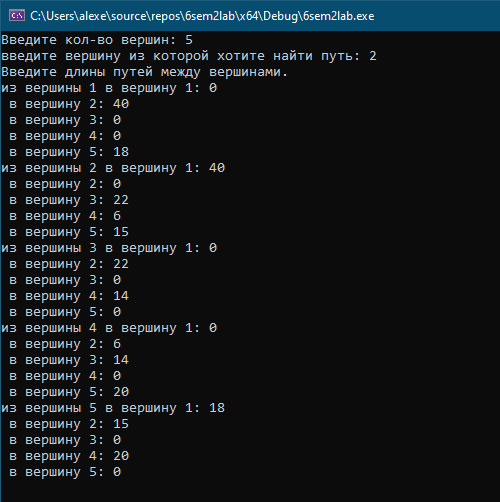


Рисунок 4 — ввод значений графа в программу

После ввода последнего значения алгоритм просчитает все пути из стартовой вершины до остальных и выведет их значения:

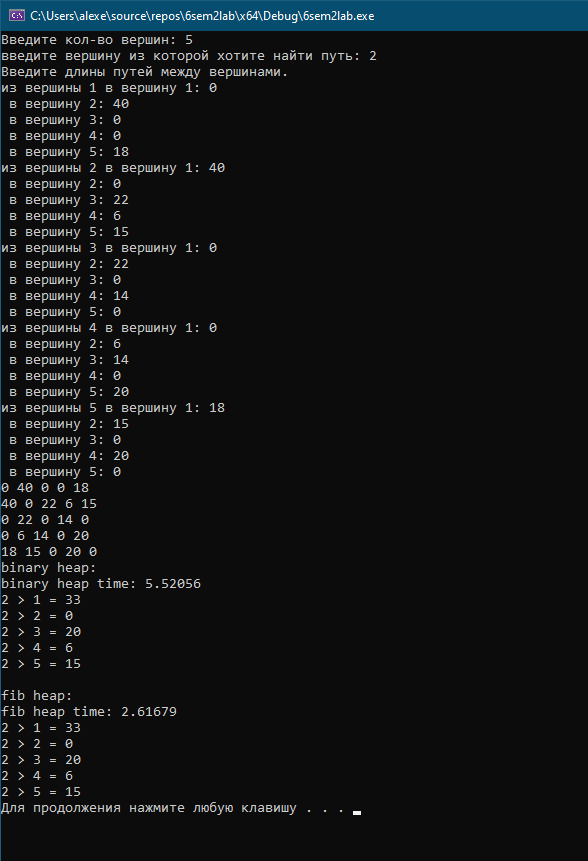


Рисунок 5 — вывод длин путей до вершин

Как видно, у нас совпали все пути, а значит алгоритм работает верно. Вот ещё пример для рандомного сильно связного графа из 20 вершин. Наш граф в виде матрицы:

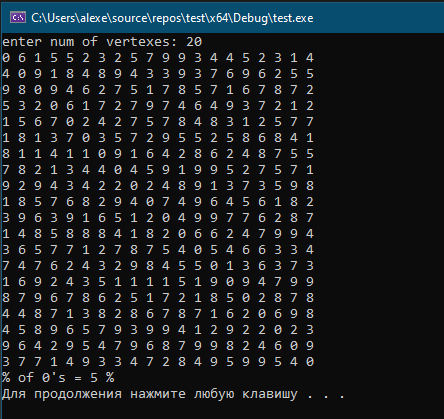


Рисунок 6 — матрица рандомного графа из 20 вершин

А вот его длины путей из 17 вершины:

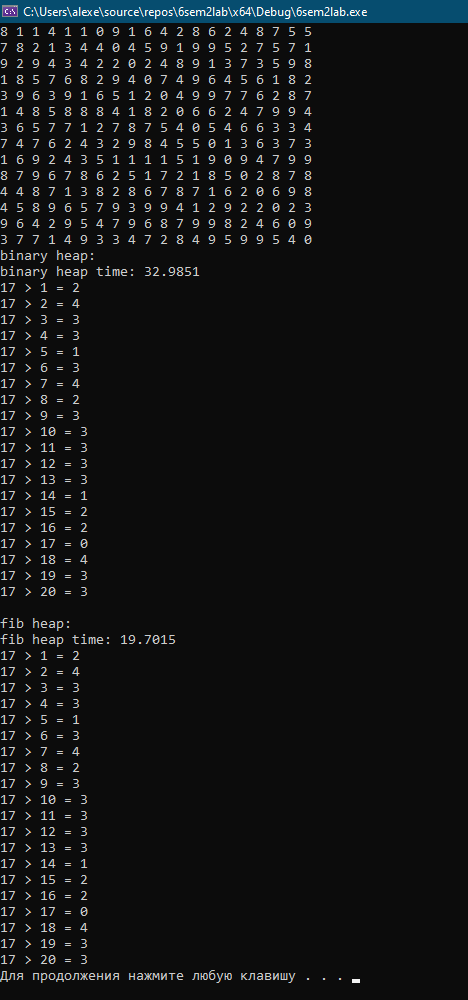


Рисунок 7 — длины путей из 17 вершины до остальных вершин

Длины одинаковые для реализаций алгоритма Дейкстры с обеими кучами, значит обе кучи реализованы правильно, как и сам алгоритм (в этом мы убедились ещё на прошлом примере).

**Вывод**

В ходе выполнения данной лабораторной работы были реализованы две кучи: двоичная и Фибоначчиева. А также реализован алгоритм Дейкстры, который нужен для нахождения кратчайших путей (если они есть) из одной вершины графа до остальных. Реализация алгоритма с Фибоначчиевой кучей показала себя быстрее, за счёт того, что вставка элемента в кучу производится за O(1), в отличие от двоичной кучи, где это происходит за O(log(n)), где n это количество элементов в куче. Сами кучи ведут себя одинаково для графов низкой/средней/высокой связности. Для низкой связности логично низкое время работы, так как при отсутствии большого количества путей будет меньше шансов, что потребуется обновлять расстояние до вершины, следовательно, не нужно будет вставлять элемент в кучу, экономя кучу времени (\*BA DUM TSS\*). Для графа высокой связности похожая ситуация, много путей, и практически из каждой вершины есть прямой путь до любой другой вершины, и он наверняка окажется короче, чем сумма путей, если идти через другие вершины, а не напрямую. Следовательно, опять мало шансов, что надо обновлять путь и вставлять новое значение в кучу. Но всё-таки такие случаи могут быть и встречаются чаще, чем для графа низкой связности, так что время на обработку нужно будет больше. И самое интересное граф средней связности, скорее всего самый часто встречающийся граф и для него времени на обработку нужно больше всего, так как отсутствие прямых путей из одной вершины в другую заставляет искать другой путь, а наличие приличного количества путей не даст так просто найти самый короткий.

Ссылка   
<https://github.com/FOOZBY/6sem2lab.git>

<https://github.com/FOOZBY/test.git> — программа для создания матрицы графа, если захотите попробовать алгоритм Дейкстры на больших графах.