**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: «Алгоритмы на графах»**

Студентка гр. 6381 \_\_\_\_\_\_\_\_\_\_\_\_\_ Лопатина А.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Филатов А.Ю.

Санкт-Петербург

2018

**Задание 1.**

Разработайте программу, которая решает задачу построения пути в графе при помощи **жадного алгоритма**. Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

В первой строке через пробел указываются начальная и конечная вершины  
Далее в каждой строке указываются ребра графа и их вес.  
В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

**Описание алгоритма.**

1. На вход подаются начальная и конечная вершина, а затем по 2 вершины и вес между ними. На основе входных данных строится матрица смежности.
2. Пока не достигли конечной вершины выбираем наименьший имеющийся путь из текущей вершины в соседнюю и идем по нему. При этом если в графе имеется цикл, то программа войдет в него, если этот путь по стоимости будет меньше, но не зациклится, так как посещенные ребра учитываются.
3. Как только дошли до конечной вершины, полученный путь выводится.

**Тестирование.**

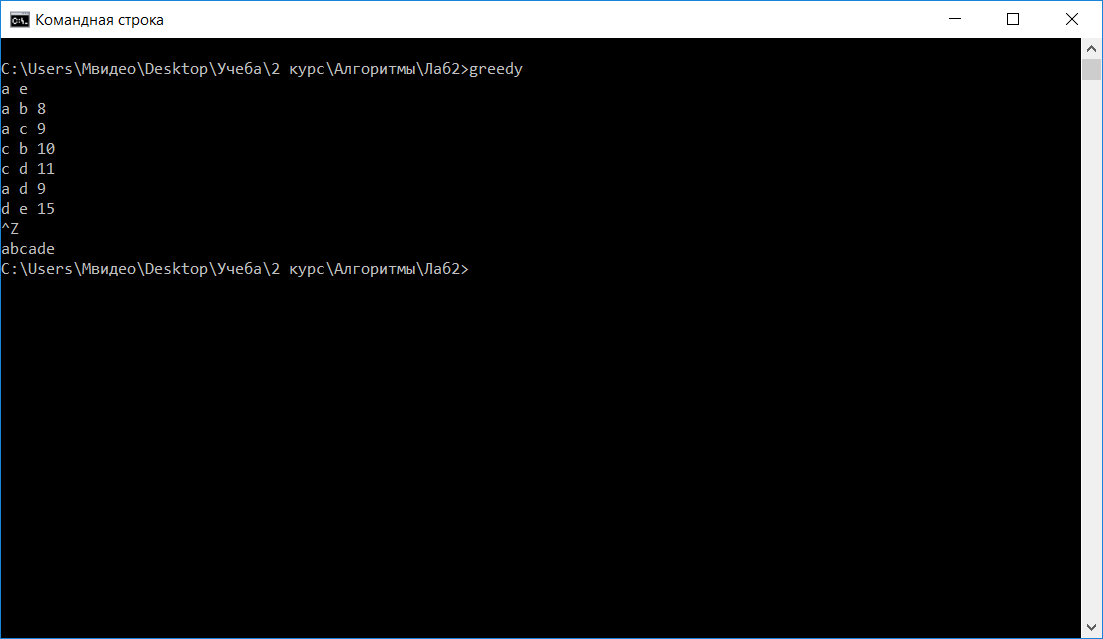


рис. 1

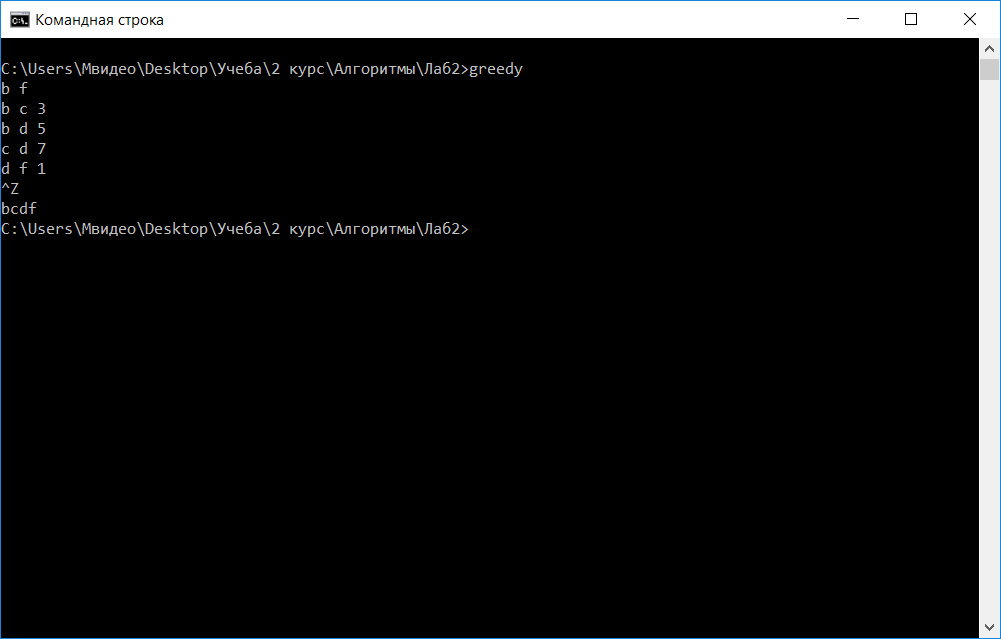


рис. 2

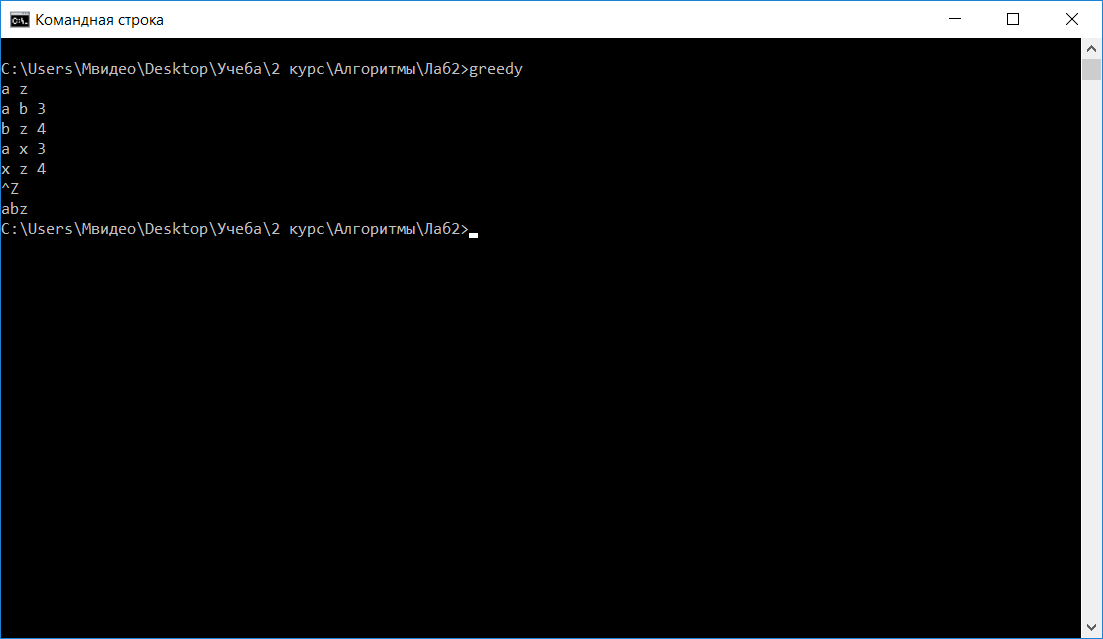


рис. 3

**Задание 2.**

Разработайте программу, которая решает задачу построения кратчайшего пути в графе **методом А\***. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции следует взять близость символов, обозначающих вершины графа, в таблице ASCII.

В первой строке через пробел указываются начальная и конечная вершины. Далее в каждой строке указываются ребра графа и их вес.  
В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

**Описание алгоритма.**

1. На вход подаются начальная и конечная вершина, а затем по 2 вершины и вес между ними. На основе входных данных строится список смежности.
2. Создаем приоритетную очередь, в которой будем хранить вершины, в которые можно пойти и их приоритет. Пока очередь не пустая, будем брать из нее вершины и рассматривать их соседей.
3. Приоритетом является сумма пути от начальной вершины и эвристической функции. Среди вершин в очереди следующей мы выбираем вершину с наименьшим приоритетом и делаем ее текущей. Как только программа достигнет конечной вершины, она выйдет из цикла.
4. Из полученного ассоциативного массива в обратном порядке формируется путь.

**Описание функций и структур данных.**

**Описание функций:**

1. **void search\_path (Vertex first, Vertex last)** – основная функция поиска пути;
2. **double heuristic(Pair temp, Vertex last)**– эвристическая функция;

**Описание структур данных:**

class Pair {

public:

Pair(char name, double weight): name(name), weight(weight) {};

char name = 0;

double weight = 0;

};

Класс Pair представляет собой класс пар, в котором хранятся данные об имени вершины, в которую есть путь из данной вершины и о весе между ними.

class Vertex {

public:

Vertex(char name) : name(name){};

list<Pair> list\_edge;

char get\_name() const {};

private:

char name;

};

Класс Vertex представляет собой класс вершин. Он содержит в себе имя вершины name, а также список list\_edge всех возможных путей, в которые ведет данная вершина и вес между ней и соседом. Для использования имени вершины реализован геттер get\_name().

class Graph {

public:

Graph() {}

vector<Vertex> list\_vertex;

map <char, char> came\_from;

map <char, double> cost\_to\_top;

Vertex& at(char a){};

void search\_path (Vertex first, Vertex last);

double heuristic(Pair temp, Vertex last) {}ж

};

Класс Graph является классом графа. Он содержит вектор list\_vertex из вершин, которые представляют собой классы, ассоциативный массив came\_from, в котором хранятся пары из вершин: из которой ушли и в которую пришли и ассоциативный массив cost\_to\_top, в котором хранятся пары из вершины и стоимости пути до нее из начальной вершины. Реализована функция at для получения ссылки на элемент list\_vertex и описаны 2 функции: search\_path и heuristic;

class Pair\_prior {

public:

Pair\_prior(Vertex vert, double prior): vert(vert), prior(prior){};

Vertex vert;

double prior;

bool operator<(const Pair\_prior b) const {};

};

Класс Pair\_prior реализован для приоритетной очереди и содержит в себе вершину vert и ее приоритет prior. Для сортировки очереди в порядке возрастания реализован оператор «<».

**Тестирование.**

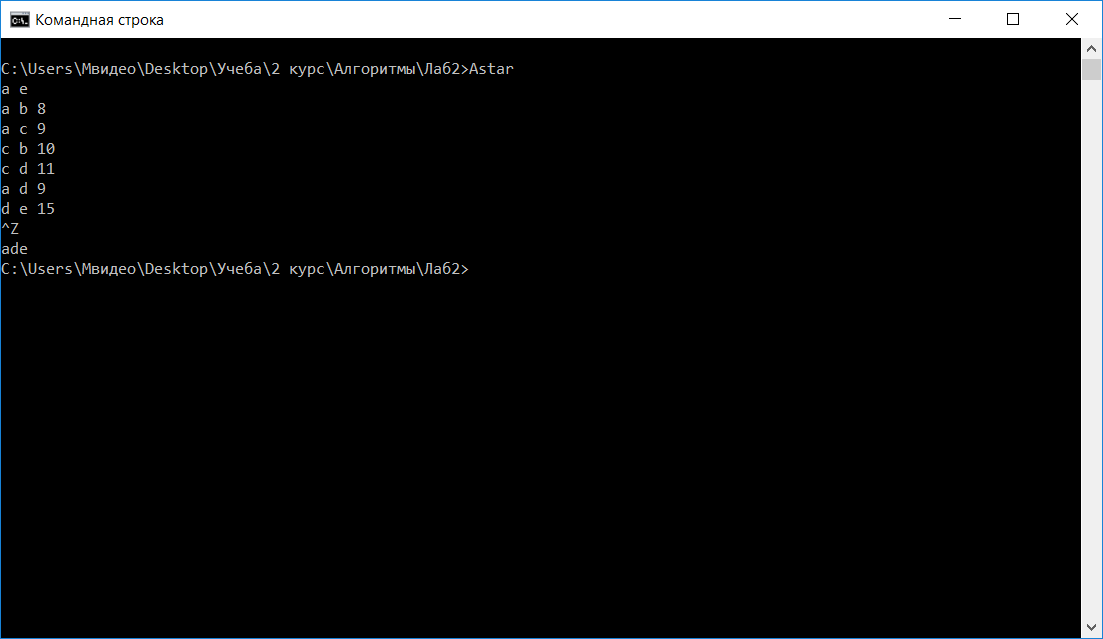


рис. 4

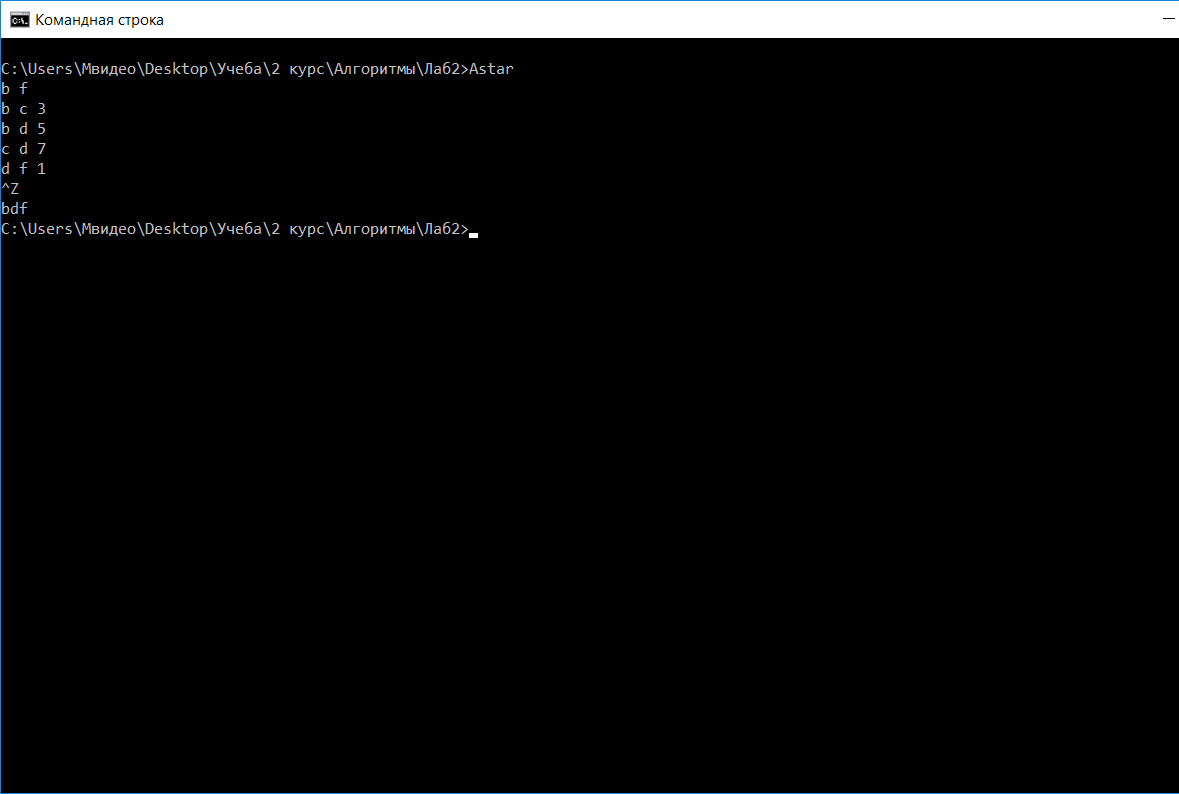


рис. 5

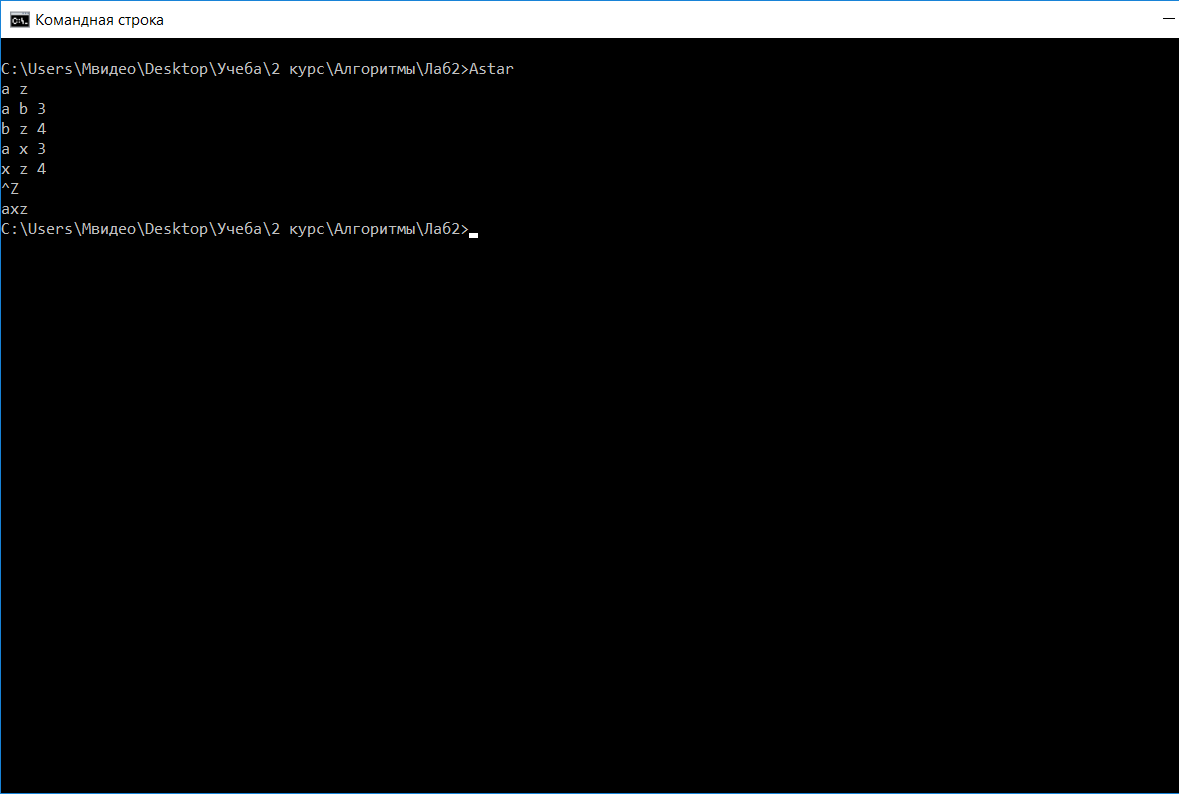


рис. 6

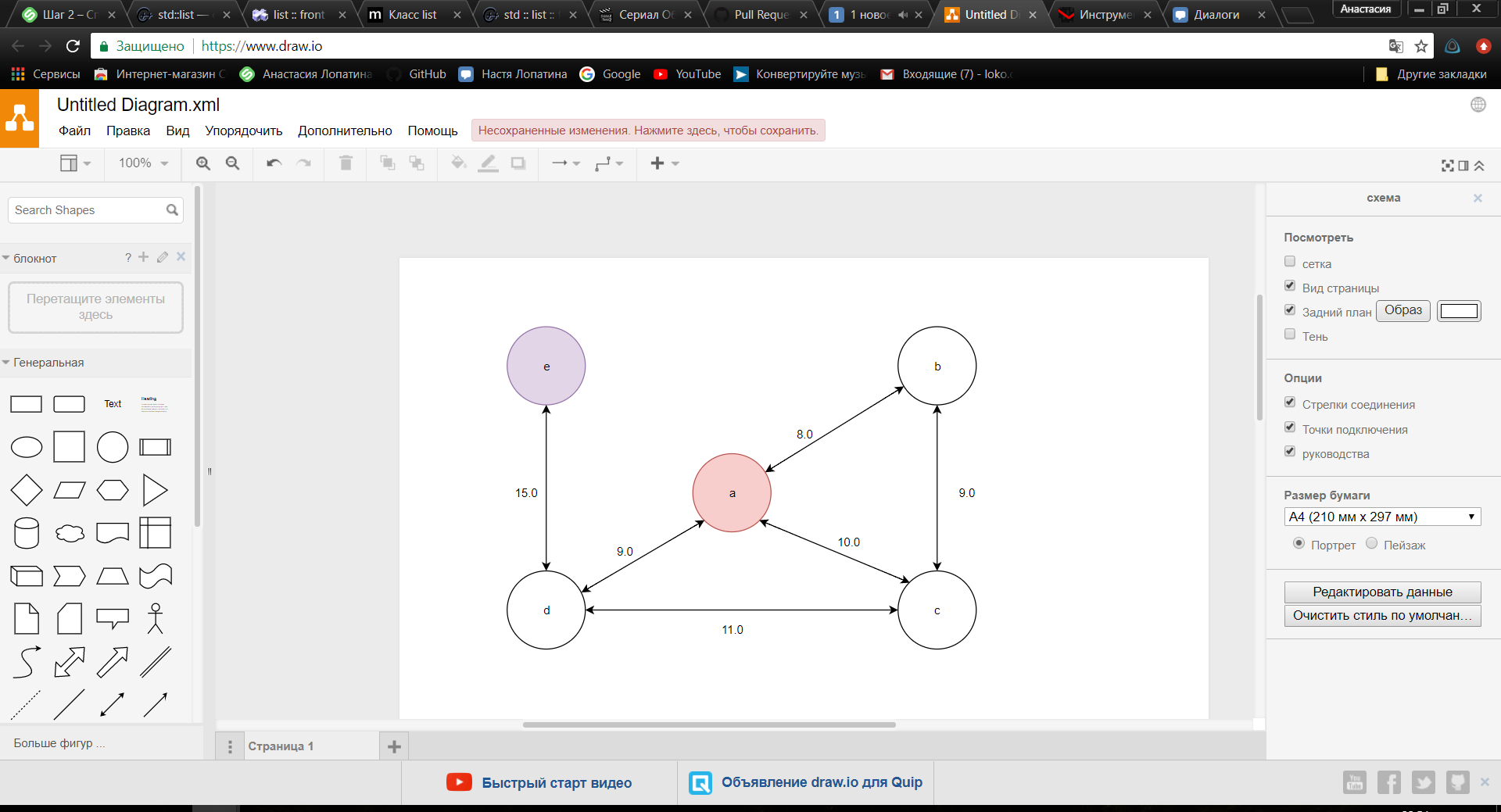


рис. 7 Граф к тесту 1

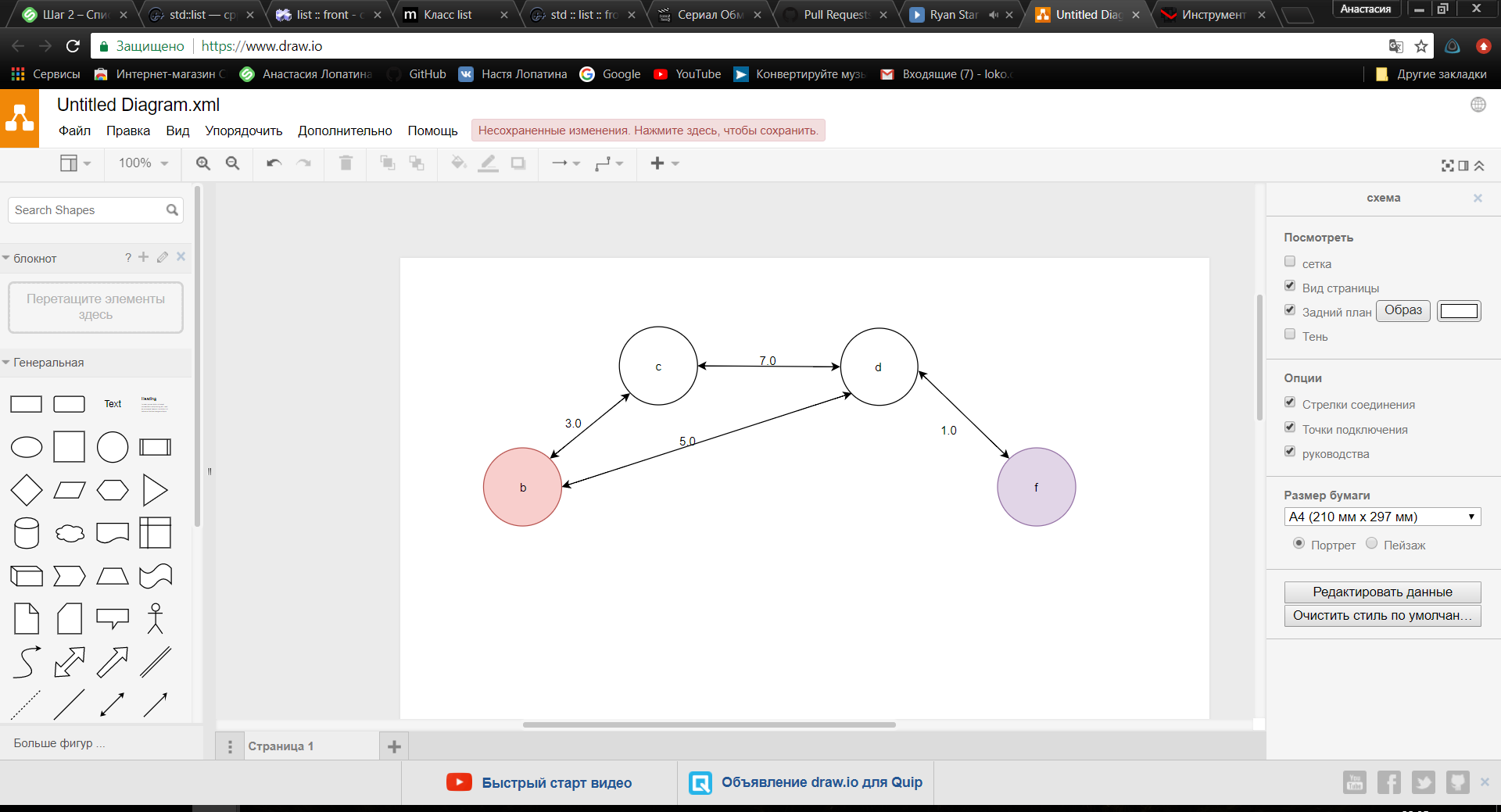


рис. 8 Граф к тесту 2

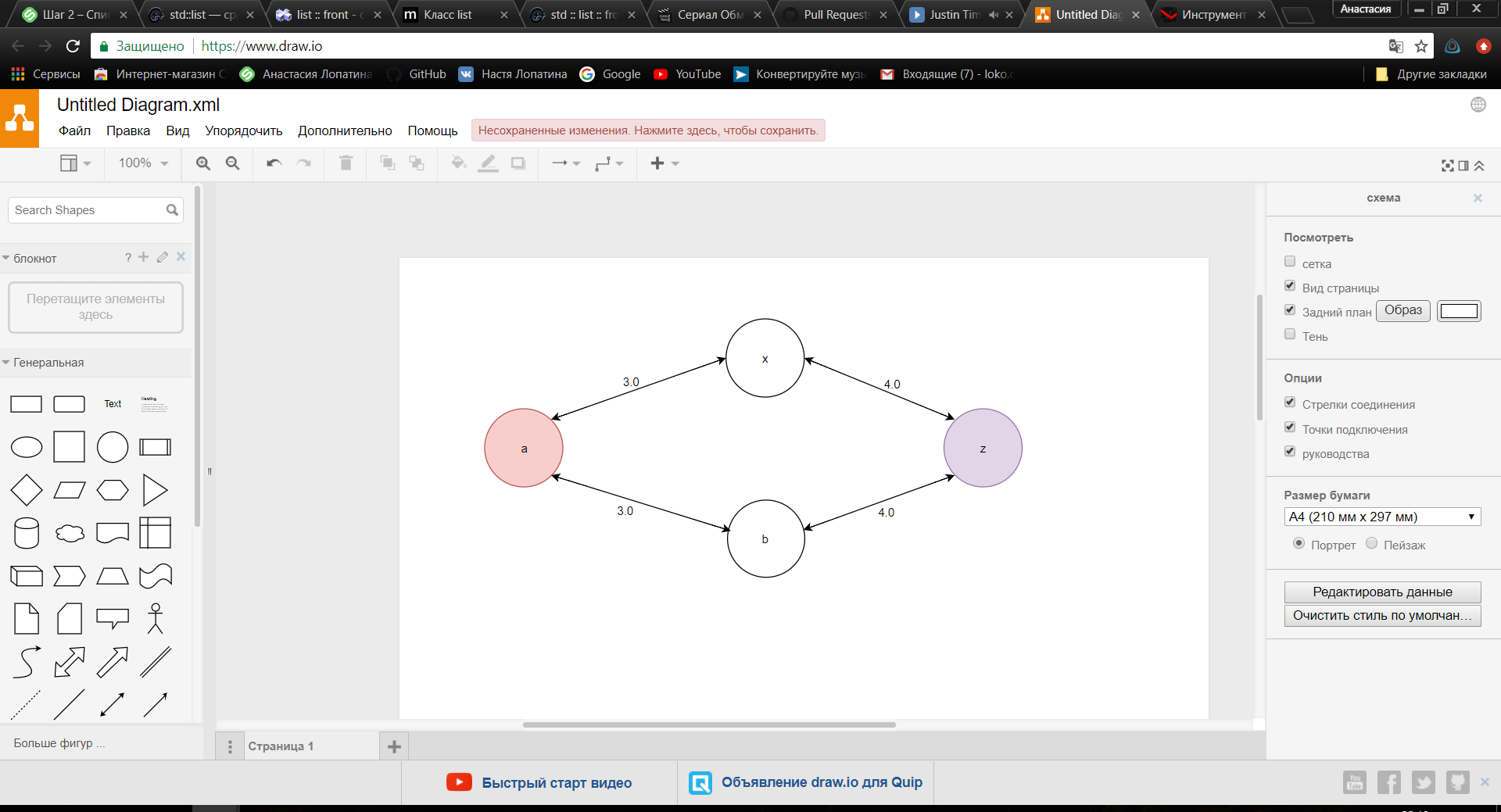


рис. 9 Граф к тесту 3

**Выводы.**

В ходе лабораторной работы мы подробно ознакомились с жадным алгоритмом и алгоритмом А стар. В результате выполнения лабораторной работы были разработаны программы, которые вычисляют кратчайший путь в неориентированном графе разными способами.