МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

доцент		Н. А. Волкова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Сетевые модели

по дисциплине: Прикладные модели оптимизации

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	Z1431		М.Д.Быстров
	номер группы	подпись, дата	инициалы, фамилия

Студенческий билет № 2021/3572

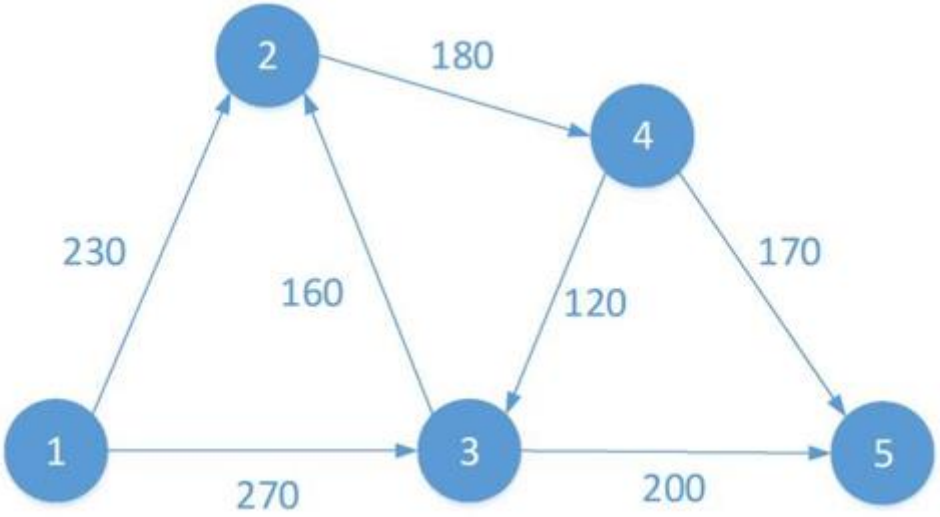
Санкт-Петербург 2025

Оглавление

Задание

Задание

На рисунке показана транспортная сеть, состоящая из пяти городов (расстояния между городами (в км) приведены возле соответствующих дуг сети). Необходимо найти кратчайшие расстояния от города 1 (узел 1) до всех остальных четырех городов двумя методами (Дейкстры и Флойда).



Результат выполнения программы

Работа выполнена в виде программы на языке программирования Python. Ниже представлены шаги работы программы при решении заданного варианта.

Матрица смежности

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 230. 270. 0. 0.]
 [0. 0. 0. 0. 180. 0.]
 [0. 0. 160. 0. 0. 200.]
 [0. 0. 0. 120. 0. 170.]
 [0. 0. 0. 0. 0. 0.]]

Алгоритм Дейкстры

Маршруты из точки 1:

Из 1 в 2: длина 230.0, маршрут [1, 2]

Из 1 в 3: длина 270.0, маршрут [1, 3]

Из 1 в 4: длина 410.0, маршрут [1, 2, 4]

Из 1 в 5: длина 470.0, маршрут [1, 3, 5]

Алгоритм Флойда

Таблица расстояний:

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 230. 270. 410. 470.]
 [0. inf 0. 300. 180. 350.]
 [0. inf 160. 0. 340. 200.]
 [0. inf 280. 120. 0. 170.]
 [0. inf inf inf inf 0.]]

Маршруты из точки 1:

Из 1 в 2: длина 230.0, маршрут [1, 2]

Из 1 в 3: длина 270.0, маршрут [1, 3]

Из 1 в 4: длина 410.0, маршрут [1, 2, 4]

Из 1 в 5: длина 470.0, маршрут [1, 3, 5]

Исходный код

Файл “lab3.py”

```
import numpy as np;

# алгоритм Дейкстры
def dijkstra(paths, src, dest):

    # все вершины
    vertexes = set(range(1, paths.shape[0]));

    vertex_num = len(vertexes);

    # расстояние и путь от вершины до источника
    ways = dict();
    routes = dict();

    # для источника расстояние - 0,
    # для остальных - бесконечность
    for vertex in vertexes:
        if (vertex == src):
            ways[vertex] = 0;
            routes[vertex] = [src];
        else:
            ways[vertex] = float("inf");

    # посещенные вершины
    visited = set();

    # пока посещены не все вершины
    while (len(visited) < len(vertexes)):

        # непосещенные вершины
        not_visited = vertexes - visited;

        min_vertex = -1;
        min_way = float("inf");

        for vertex, way in ways.items():
            if (way < min_way and vertex in not_visited):
                min_way = way;
                min_vertex = vertex;

        if (min_vertex == -1): raise Exception("Минимальная вершина не найдена");

        vertex = min_vertex;
        way = min_way;

        # для всех непосещенных соседей вычисляем расстояние и путь
```

```
        for neighbour in range(1, vertex_num  + 1):
            if (paths[vertex, neighbour] != 0 # есть путь
                and neighbour not in visited # сосед не был посещен
                and way + paths[vertex, neighbour] < ways[neighbour]): # путь через
текущую вершину лучше

                ways[neighbour] = way + paths[vertex, neighbour];
                routes[neighbour] = routes[vertex] + [neighbour];

        visited.add(vertex);

    # если расстояние и маршрут до назначения найден, вернем его
    # иначе вернем -1
    if dest in ways and dest in routes:
        return [ways[dest], routes[dest]];
    else:
        return [-1, []];

# алгоритм Флойда
def floyd(paths):

    d = paths.copy();
    next = np.zeros(paths.shape);
    vertex_num = d.shape[0] - 1;
    vertexes = set(range(1, vertex_num + 1));
    routes = dict();

    for u in vertexes:
        for v in vertexes:

            # длина ребра между вершинами
            d[u][v] = paths[u][v]

            # нет пути - бесконечность
            if (d[u][v] == 0): d[u][v] = float("inf");

            # следующая вершина по прохождению к v через u
            next[u][v] = v;

    # для каждой вершины путь = 0,
    # до себя ближе всего через себя
    for v in vertexes:
        d[v][v] = 0;
        next[v][v] = v;

    for i in vertexes:
        for u in vertexes:
            for v in vertexes:

                # если от u до v короче через i
```

```
                # запишем что надо ходить через i
                if d[u][i] + d[i][v] < d[u][v]:
                    d[u][v] = d[u][i] + d[i][v];

                    # отправим из u в i вместо того чтобы идти напрямую в v
                    next[u][v] = next[u][i];

    # рассчитаем для всех вариантов маршруты
    for u in vertexes:
        for v in vertexes:
            if d[u][v] == 0: continue;

            c = u;

            routes[(u, v)] = [];

            while  c != v:
                routes[(u, v)].append(c);
                c = int(next[c][v]);

            routes[(u, v)].append(c);


    return [d, routes];


# кол-во вершин
vertex_num = 5;

# матрица смежности графа
# 0 - нет пути
# сделаем индексацию с 1 для упрощения чтения и понимания
paths = np.zeros((vertex_num + 1, vertex_num + 1), float);

paths[1][2] = 230;
paths[1][3] = 270;
paths[2][4] = 180;
paths[3][2] = 160;
paths[3][5] = 200;
paths[4][3] = 120;
paths[4][5] = 170;

print("Матрица смежности");
print(paths);

print("Алгоритм Дейкстры");
print("Маршруты из точки 1:");

# начальный пункт всегда 1
src = 1;
```

```
for dest in range(2, vertex_num + 1):
    [way, route] = dijkstra(paths, src, dest);
    print("Из " + str(src) + " в " + str(dest) + ": длина " + str(way) + ", маршрут " +
str(route));

print("Алгоритм Флойда");

[d, routes] = floyd(paths);

print("Таблица расстояний:");
print(d);

# начальный пункт всегда 1
src = 1;

print("Маршруты из точки 1:");

for dest in range(2, vertex_num + 1):
    way = d[src][dest];
    route = routes[(src, dest)] if (src, dest) in routes else [];

    [way, route] = dijkstra(paths, src, dest);
    print("Из " + str(src) + " в " + str(dest) + ": длина " + str(way) + ", маршрут " +
str(route));
```

Вывод

В ходе выполнения третьей лабораторной работы создана программа на языке Python для нахождения кратчайшего пути из одной точки графа ко всем остальным.

Изучены алгоритмы для нахождения кратчайшего пути в графе.