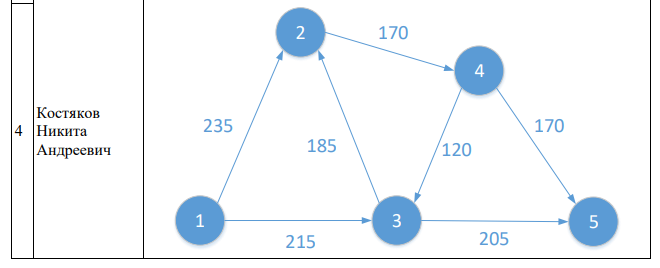
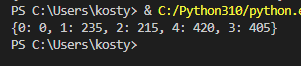
Практическое задание 5

Сетевые модели

**Задание.** На рисунке показана транспортная сеть, состоящая из пяти городов (расстояния между городами (в милях) приведены возле соответствующих дуг сети). Необходимо найти кратчайшие расстояния от города 1 (узел 1) до всех остальных четырех городов двумя методами (Дейкстры и Флойда). Теория и примеры разобраны в лекции 8.



## Дейкстры



def dijkstra\_shortest\_path(graph, start, p={}, u=[], d={}):

    if len(p) == 0: p[start] = 0 # инициализация начального пути

    # print "start V: %d, " % (start)

    for x in graph[start]:

        if (x not in u and x != start):

            if (x not in p.keys() or (graph[start][x] + p[start]) < p[x]):

                p[x] = graph[start][x] + p[start]

    u.append(start)

    min\_v = 0

    min\_x = None

    for x in p:

        # print "x: %d, p[x]: %d, mv %d" % (x, p[x], min\_v)

        if (p[x] < min\_v or min\_v == 0) and x not in u:

                min\_x = x

                min\_v = p[x]

    if(len(u) < len(graph) and min\_x):

        return dijkstra\_shortest\_path(graph, min\_x, p, u)

    else:

        return p

if \_\_name\_\_ == '\_\_main\_\_':

    # инициализация графа с помощью словаря смежности

    a, b, c, d, e = range(5)

    N = [

        {b: 235, c: 215},

        {d: 170},

        { b: 186, e: 205},

        {b: 15, c: 120, e: 170},

        {}

    ]

    for i in range(1):

        print (dijkstra\_shortest\_path(N, a))

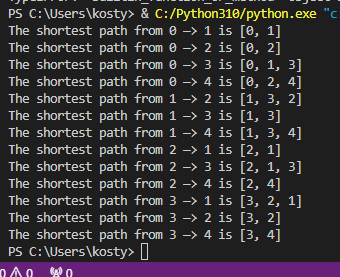
# b in N[a] - смежность

# len(N[f]) - степень

# N[a][b] - вес (a,b)

# print N[a][b]

## Флойда



# Рекурсивная функция для печати пути заданной вершины `u` из исходной вершины `v`

def printPath(path, v, u, route):

    if path[v][u] == v:

        return

    printPath(path, v, path[v][u], route)

    route.append(path[v][u])

# Функция для печати кратчайшей стоимости с путем

# Информация # между всеми парами вершин

def printSolution(path, n):

    for v in range(n):

        for u in range(n):

            if u != v and path[v][u] != -1:

                route = [v]

                printPath(path, v, u, route)

                route.append(u)

                print(f'The shortest path from {v} —> {u} is', route)

# Функция для запуска алгоритма Флойда-Уоршалла

def floydWarshall(adjMatrix):

    # Базовый вариант

    if not adjMatrix:

        return

    # общее количество вершин в `adjMatrix`

    n = len(adjMatrix)

    # Матрица стоимости и пути # хранит кратчайший путь

    # Информация # (кратчайшая стоимость/кратчайший маршрут)

    # изначально, стоимость будет равна весу лезвия

    cost = adjMatrix.copy()

    path = [[None for x in range(n)] for y in range(n)]

    # инициализирует стоимость и путь

    for v in range(n):

        for u in range(n):

            if v == u:

                path[v][u] = 0

            elif cost[v][u] != float('inf'):

                path[v][u] = v

            else:

                path[v][u] = -1

    # запускает Флойда-Уоршалла

    for k in range(n):

        for v in range(n):

            for u in range(n):

                # Если вершина `k` находится на кратчайшем пути из `v` в `u`,

                #, затем обновите значение cost[v][u] и path[v][u]

                if cost[v][k] != float('inf') and cost[k][u] != float('inf') \

                        and (cost[v][k] + cost[k][u] < cost[v][u]):

                    cost[v][u] = cost[v][k] + cost[k][u]

                    path[v][u] = path[k][u]

            #, если диагональные элементы становятся отрицательными,

            # Graph # содержит цикл отрицательного веса

            if cost[v][v] < 0:

                print('Negative-weight cycle found')

                return

    # Вывести кратчайший путь между всеми парами вершин

    printSolution(path, n)

if \_\_name\_\_ == '\_\_main\_\_':

    # определить бесконечность

    I = float('inf')

    # с учетом представления смежности матрицы

    graph=[[I, 235, 215, I, I],

            [I, 0, I, 170, I ],

            [I, 185, 0, I, 205],

            [I,I,120,0,170],

            [I,I,I,I,0]]

    # Запустите алгоритм Флойда-Уоршалла

    floydWarshall(graph)