**Отчет**

Выполнил студент Шакирзянов Руслан 4214

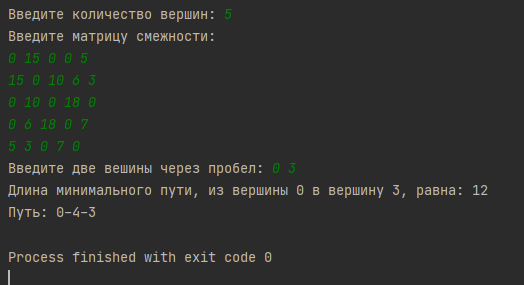
**Задача**

|  |  |  |
| --- | --- | --- |
| Построение дерева достижимости и определения кратчайшего пути | Одна из форм представления графов (матрица смежности, матрица инциденций, списки смежности) – по выбору | Кратчайший путь между двумя вершинами |

**Процедурное решение**

Так как в программе нахождения всех простых цепей уже используется аналог дерева достижимости используем эту функцию для нахождения кратчайшего пути между двумя вершинами. Находим все простые цепи и выбираем из них те чье начало и конец совпадают с заданными вершинами. Проходимся по всем выбранным цепям и при помощи списка ребер находим их длину, беря по парно вершины цепи, сравниваем минимально найденной длиной цепи и если найдено меньше то записываем новую минимальную дину и путь. По завершению цикла буде найден минимальный путь между двумя заданными вершинами.

**Пример работы программы**



Код программы

def SmejToSmejList(vertex\_num, smejnost\_matrix):  
 smejnost\_lists = [[i] for i in range(vertex\_num)] # создание списка смежности с номерами вершин  
 for i in range(len(smejnost\_matrix)):  
 for j in range(len(smejnost\_matrix[i])): # проход по матрице смежности  
 if smejnost\_matrix[i][j] != 0: # если найдена смежная вершина  
 smejnost\_lists[i].append([j, smejnost\_matrix[i][j] if smejnost\_matrix[i][j] != 2 else 1]) # добавить в список смежности пару в формате [вершина, вес]  
 return smejnost\_lists  
  
  
def SimpleCep(vertex\_start, length, smejnost\_list, dict\_of\_chains, vertex\_num): # функция нахождения простых цепей  
 chain = [vertex\_start] # масив депи с изначально заданной ставтовой вершиной  
 chains = [] # массив всех цепей  
 if (vertex\_start, length) in dict\_of\_chains: # если кортеж из стартовой вершины и данной длинны уже есть в словаре цепей то вывести его  
 return dict\_of\_chains[(vertex\_start, length)]  
 if length == 1: # добавление цепей в список если длинна цепи равна 1 (фактически переписывание списка смежности для каждой вершины в удобном формате)  
 for i in range(vertex\_num):  
 if i in smejnost\_list[vertex\_start] and vertex\_start != i:  
 for j in range(smejnost\_list[vertex\_start].count(i)):  
 chain.append(i)  
 chains.append(chain)  
 chain = [vertex\_start]  
 dict\_of\_chains[(vertex\_start, length)] = chains # добавление цепей длинны 1 в словарь цепей  
 return chains  
 else:  
 for i in range(vertex\_num): # проход по всем вершинам  
 if i in smejnost\_list[vertex\_start] and vertex\_start != i: # проверка на то что статовая вершина не равна i для избежания лишних операций  
 for j in range(smejnost\_list[vertex\_start].count(i)):  
 for k in SimpleCep(i, length - 1, smejnost\_list, dict\_of\_chains, vertex\_num): # рекурсивный вызов функции с уменьшением длины цепей  
 if vertex\_start not in k:  
 chain = [vertex\_start] + k  
 chains.append(chain)  
 chain = []  
 dict\_of\_chains[(vertex\_start, length)] = chains # добавление цепей в словарь цепей  
 return chains  
  
  
def ToEdgeList(smejnost\_list): # функция создания словаря ребер из списков смежности с с элементами по индексам (ребро1б ребро2) и значениями весов  
 edge\_list = dict()  
 for i in range(len(smejnost\_list)):  
 for j in smejnost\_list[i]:  
 if (i, j[0]) not in edge\_list and (j[0], i) not in edge\_list:  
 edge\_list[(i, j[0])] = j[1]  
 return edge\_list  
  
  
def ToNevzvesh(smejnost\_list): # функция перевода взвешенных списков смежности в невзвешанные  
 out = []  
 for i in smejnost\_list:  
 temp\_list = []  
 for j in i:  
 temp\_list.append(j[0]) # преобразование пары (вершина, вес) в просто элемент вершина  
 out.append(temp\_list)  
 return out  
  
  
def MinWay(dict\_of\_chains, edge\_list, start\_vertex, end\_vertex):  
 ways = [] # список путей  
 key\_list = dict\_of\_chains.keys() # список ключей словаря простых цепей  
 for i in key\_list:  
 for j in dict\_of\_chains[i]: # проход по всем цепям  
 if j[0] == start\_vertex and j[-1] == end\_vertex: # если начальная и конечная вершины совпадают с заданными  
 ways.append(j) # вводим эту цепь в список путей  
 min\_way = (-1, [-1]) # минимальный путь  
 for way in ways: # проход по всем путям найденным выше  
 len\_way = 0 # длина пути  
 for i in range(len(way) - 1): # проходим по вершинам пути и через словарь ребер добавляем их вес в длину пути  
 if (way[i], way[i+1]) in edge\_list: # т.к. в словаре ребро указно в одном направлении проверяем оба направления  
 len\_way += edge\_list[(way[i], way[i+1])]  
 else:  
 len\_way += edge\_list[(way[i+1], way[i])]  
 if min\_way[0] == -1 or min\_way[0] > len\_way: # если путь окажется меньше чем найденный ранее минимум  
 min\_way = (len\_way, way) # то записываем длинну пути и сам путь в минимум  
 print(f"Длина минимального пути, из вершины {start\_vertex} в вершину {end\_vertex}, равна: {min\_way[0]}")  
 print("Путь:", "-".join(str(i) for i in min\_way[1])) # красивыйй вывод минимального пути  
  
  
def main():  
 vertex\_num = int(input("Введите количество вершин: ")) # запрос навведенение количества вершин  
 print("Введите матрицу смежности:")  
 smejnost\_matrix = [[int(i) for i in input().split()] for j in  
 range(vertex\_num)] # ввод матрицы смежности с клавиатуры  
 smejnost\_list\_temp = SmejToSmejList(vertex\_num, smejnost\_matrix)  
 smejnost\_list\_v = [] # создание взвешанного списка смежности  
 for i in range(vertex\_num):  
 temp\_list = []  
 for j in range(1, len(smejnost\_list\_temp[i])):  
 temp\_list.append(smejnost\_list\_temp[i][j]) # изменение структуры списка смежности из предыдущих работ  
 smejnost\_list\_v.append(temp\_list)  
 edge\_list = ToEdgeList(smejnost\_list\_v) # создание списа ребер  
 smejnost\_list\_nv = ToNevzvesh(smejnost\_list\_v) # создание невзвешанного списка смености  
 simple\_cep\_list = [] # список простых цпей  
 dict\_of\_chains = dict() # словарь цепей  
 for i in range(vertex\_num):  
 for j in range(vertex\_num - 1, 1, -1):  
 simple\_cep\_list.append(SimpleCep(i, j, smejnost\_list\_nv, dict\_of\_chains, vertex\_num)) # заполнение списка простых цепей  
 start\_vertex, end\_vertex = [int(i) for i in input("Введите две вешины через пробел: ").split()] # ввод вершин между которыми надо найти минимальный путь  
 MinWay(dict\_of\_chains, edge\_list, start\_vertex, end\_vertex)  
  
  
main()  
""" пример  
5  
0 15 0 0 5  
15 0 10 6 3  
0 10 0 18 0  
0 6 18 0 7  
5 3 0 7 0  
0 3  
"""