**Федеральное государственное бюджетное образовательное учреждение высшего образования «Национальный исследовательский университет «МЭИ»**

Институт информационных и вычислительных технологий

Кафедра математического моделирования

Дисциплина «Технологии программирования»

Отчет к лабораторной работе № 2

"Разработка программы для работы с графом на языке Python"

Выполнил:

Симаков Александр Максимович

Принял:

Доцент Князев А.В.

Москва

2022 год

**Задание:**

Найти самый длинный путь из одной вершины в другую, не проходящий через два заданных ребра в определенном направлении. (граф\_спис)

**Описание работы программы:**

Вводим граф из файла *txt* в виде массива массивов. Индекс каждого массива внутри него - это вершина, а сам массив - это список смежных с этой вершиной сторон. Используем обход в глубину для поиска максимального пути. Проверяем, чтобы два заданных ребра в него не входили.

**Основные операции:**

* **find\_res(self, vn, vk, a1, a2, b1, b2)**

На вход получает следующие параметры: вершины, для которых нужно найти максимальный путь, и два ребра, через которые не должен проходить наш искомый путь. Функция реализована на основе поиска в глубину, приведенного в пособии. Ищет все пути из vn в vk. Если найден первый путь, удовлетворяющий условиям сохраняет его и его длину. Далее если находит новый путь, соответствующий условию, проверяет больше ли его длина, чем у предыдущего сохраненного пути, если да, сохраняет новый путь и его длину.

* **output\_res(self)**

Выводит путь-результат графически с использованием networkx, для этого преобразует исходный путь в предстваление networkx

функция *graph = nx.Graph()* создает объект графа networkx

метод *add\_node* - добавляет вершину в граф networkx

метод *add\_edge* - добавляет грань в граф networkx

* **output-gr(self)**

Выводит исходный граф графически с использованием networkx, для этого преобразует исходный путь в предстваление networkx

функция graph = nx.Graph() создает объект графа networkx

метод add\_node - добавляет вершину в граф networkx

метод add\_edge - добавляет графнь в граф networkx

**Тесты:**

* Файл input1.txt

Содержимое файла:

8

1 3 4 7

0 2 4

1 5

0

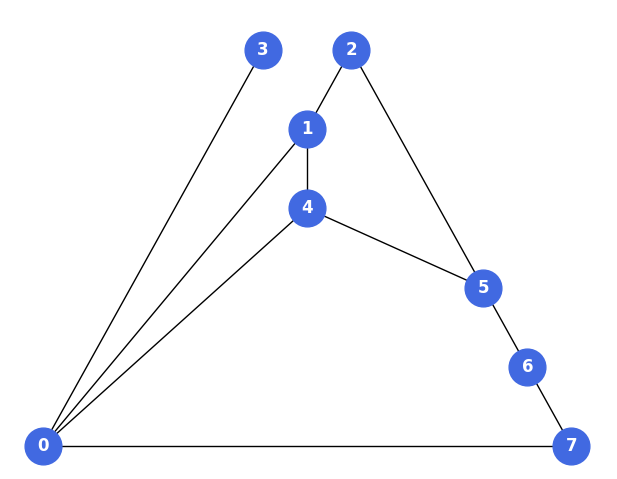
0 1 5

2 4 6

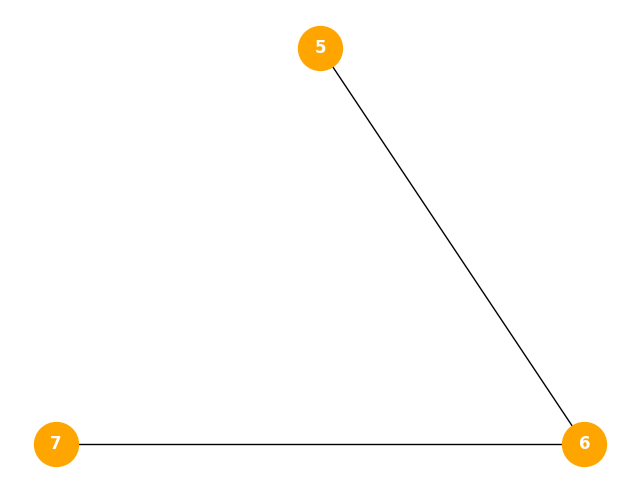
5 7

0 6

Имеем следующий граф:



Вызываем функцию gr.find\_res(7, 5, 7, 0, 0, 4). То есть, ищем максимальный путь из 7 в 5, который не проходит через рёбра 7-0 и 0-4. В результате имеем следующий путь:



Вызываем функцию gr.find\_res(7, 5, 7, 0, 7, 6). То есть, ищем максимальный путь из 7 в 5, который не проходит через рёбра 7-0 и 7-6.

Вылетает: «Граф не содержит пути, удовлетворяющего условию»

* Файл input2.txt

Содержимое файла:

10

1 2

0 3

0 4

1 4 5

2 3 6 8

3 8

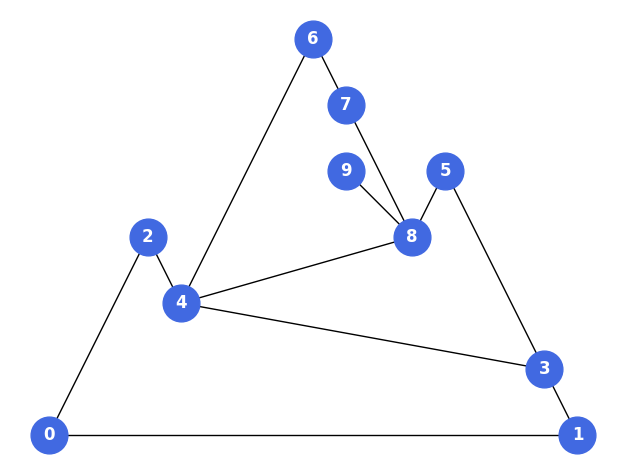
4 7

6 8

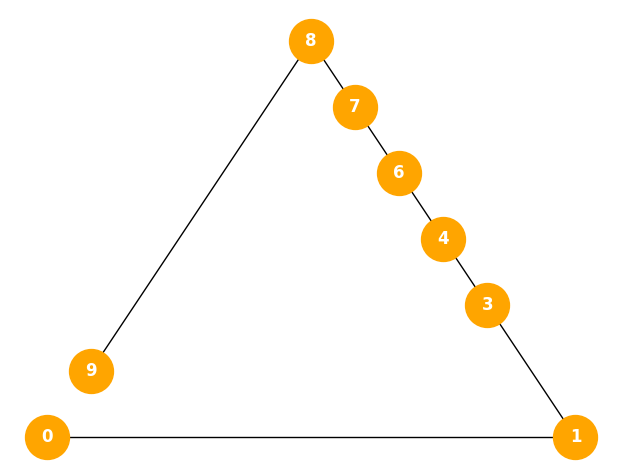
4 7 9

8

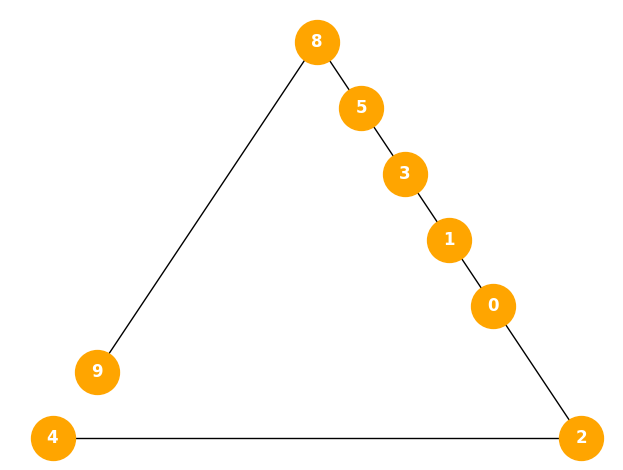
Имеем следующий граф:



Вызываем функцию gr.find\_res(0, 9, 0, 2, 4, 8). То есть, ищем максимальный путь из 0 в 9, который не проходит через рёбра 0-2 и 4-8. Получаем путь:



Вызываем функцию gr.find\_res(4, 9, 7, 8, 4, 8). То есть, ищем максимальный путь из 4 в 9, который не проходит через рёбра 7-8 и 4-8. Получаем путь:



**Листинг программы:**

* main.py

**from** graph\_impl **import** graph  
  
gr = graph()  
  
gr.input(**"input2.txt"**)  
gr.output\_gr()  
*#gr.find\_res(7, 5, 7, 0, 7, 6) # тест для графа в файле intput1.txt  
#gr.find\_res(7, 5, 0, 7, 0, 4) # тест для файла в файле intput1.txt  
#gr.find\_res(0, 9, 0, 2, 4, 8) # тест для графа в файле intput2.txt*gr.find\_res(4, 9, 7, 8, 4, 8) *# тест для графа в файле intput2.txt*gr.output\_res()

* graph\_impl.py

**import** networkx **as** nx  
**import** numpy **as** np  
  
*# Класс, описывающий граф.***from** matplotlib **import** pyplot **as** plt  
  
*# Возвр. False, если путь St прогодит через ребро (a1, a2) в направлении от a1 к a2***def** exist\_ord\_edge(St, ks, a1, a2, vk):  
 res = **True** *# Если i-й элемент fr, а i+1 -й элемент to, то True* **for** i **in** np.arange(0, ks, 1): *# ks-1* **if** (St[i] == a1) **and** (St[i+1] == a2):  
 res = **False  
 if**(St[ks] == a1) **and** (vk == a2 ):  
 res = **False  
 return** res  
  
**class** graph:  
 **def** \_\_init\_\_(self): *# Constructor* self.num\_nodes = 0  
 self.gr\_list = []  
 self.was\_res = **False** *# Сообщается, что путь, соотв. условию не был найден* self.res\_path = []  
 *# Input graph from file* **def** input(self, f\_name):  
 f = open(f\_name, **"rt"**) *# open file to read* self.num\_nodes = int(f.readline()) *# first string in file should has number of nodes* self.gr\_list = []  
 **for** line **in** f:  
 self.gr\_list.append([int(x) **for** x **in** line.split()])  
 **return** 0  
 *# Проверка, являются ли вершины i, j смежными* **def** is\_smejn(self, v1, v2):  
 **return** (v2 **in** self.gr\_list[v1])  
 *# Output graph as image* **def** output\_gr(self):  
 graph = nx.Graph()  
 **for** i **in** range(self.num\_nodes): *# Add nodes* graph.add\_node(i)  
 **for** i **in** range(self.num\_nodes): *# Add edges* **for** j **in** range(self.num\_nodes):  
 **if**(self.is\_smejn(i, j)):  
 graph.add\_edge(i, j)  
 nx.draw(graph, with\_labels=**True**,  
 font\_weight=**'bold'**,  
 font\_color=**"white"**,  
 node\_color=**'RoyalBlue'**,  
 *#node\_color=color\_map,* node\_size=700,  
 pos=nx.planar\_layout(graph)  
 )  
 plt.show()  
 *# Вывести найденый путь или сообщить, что его нет.* **def** output\_res(self):  
 **if** (self.was\_res == **False**):  
 print(**"Граф не содержит пути, удовлетворяющего условию"**)  
 **else**:  
 graph = nx.Graph()  
 *# Преобразовать график из моего предстваления в NetworkX представление* **for** i **in** np.arange(0, len(self.res\_path), 1): *# Преобр. вершины.* graph.add\_node(self.res\_path[i] ) *# Для удобства на рисуемом графике вершины нумеруются с единицы.* **for** i **in** np.arange(0, len(self.res\_path) - 1, 1): *# Преобр. грани.* graph.add\_edge(self.res\_path[i], self.res\_path[i + 1])  
 options = dict(node\_color=**'red'**, node\_size=100, width=3)  
 nx.draw(graph, with\_labels=**True**,  
 font\_weight=**'bold'**,  
 font\_color=**"white"**,  
 node\_color=**'Orange'**,  
 node\_size=1000,  
 pos=nx.planar\_layout(graph) *#* )  
 plt.show()  
 *# Найти самый длинный путь из vn в vk,  
 # не проходящий через ребра (a1, a2) в напр. от а1 к а2  
 # не проходящий через ребра (b1, b2) в напр. от b1 к b2.* **def** find\_res(self, vn, vk, a1, a2, b1, b2):  
 *# self.was\_res применяется как флаг, говорящий нахождении первого пути удв. условию* self.was\_res = **False** max\_len = 0 *# Максимальная длина последнего найденного пути, удв. условию.* M = np.zeros((self.num\_nodes)) *# Массив отметок о прохождении вершин.* ks = 0 *# "Индекс" последнего элемента в стеке, изначально равен 0* St = np.empty((self.num\_nodes), int) *# Стек.* St[ks] = vn *# Помещаем начало пути в стек.* M[vn] = 1 *# Отмечаем, что вершина-начало пути пройдена* L = 0 *#* **while** ks >= 0: *# Пока стек не пуст.* v = St[ks] *# Устанавливаем текущий элемент.* Pr = 0 *# Флаг для определения было ли прерывание* **for** j **in** np.arange(L, self.num\_nodes, 1): *# Проходим по узлам графа.* **if** self.is\_smejn(v, j): *# Если вершины v, j смежные.* **if** j == vk: *# Если путь пройден.* **if** (exist\_ord\_edge(St, ks, a1, a2, vk)  
 **and** exist\_ord\_edge(St, ks, b1, b2, vk) **and** (ks)  
 **and** (max\_len < ks)):  
 max\_len = ks  
 print(**"Путь построен: "**)  
 **for** i **in** np.arange(0, ks + 1, 1):  
 print(St[i], **", "**, end=**""**)  
 print(vk, **";"**)  
 *# Сформировать массив для вывода результата, нумерация вершин с 0* res = np.empty(ks + 2, int)  
 **for** i **in** np.arange(0, ks + 1):  
 res[i] = St[i]  
 res[ks + 1] = vk  
 self.res\_path = res  
 self.was\_res = **True  
 return  
 else**: *# Если смежный узел не является концом пути.* **if** M[j] == 0: *# Если вершина не пройдена.* Pr = 1  
 **break  
 if** Pr == 1: *# Если было прерывание.* ks += 1  
 St[ks] = j *# Добаляем новый элемент в стек.* L = 0  
 M[j] = 1 *# Отмечаем вершину как пройденную* **else**: *# Если не было прерывания, то возвращаемся к предыдущей вершине* L = v + 1  
 M[v] = 0 *# Удаляем вершину из пройденных.* ks -= 1 *# Удаляем вершину из стека.*