

---

## *Общее задание*

---

Разработать класс для представления неориентированного графа и поиска

цикла или пути между двумя вершинами при заданных условиях.

Разработать программу на языке Python, реализующую конкретное задание.

Программа должна обеспечивать ввод описания графа из текстового файла.

В скобках указана форма внутреннего представления графа: граф\_матр (матрица смежности), граф\_спис (множество списков смежных вершин)

Программа должна обеспечивать представление исходного графа и результата в графическом виде.

---

## *Индивидуальное задание*

---

Найти цикл, проходящий не более чем через две вершины центра графа (граф\_матр).

---

## *Описание работы программы*

---

Создается объект класса Graph, в него вносятся данные из файла. В моем варианте – в исходном файле матрица смежности. Далее изображаем исходный графический граф с помощью библиотеки Matplotlib. После этого переходим к индивидуальному заданию, в котором сначала ищем центр графа, затем ищем цикл проходящий не более чем через две вершины центра графа. Если цикла не найдется – выводим соответствующее сообщение, в противном случае выделяем ребра графа темно-синим цветом, если они присутствуют в искомом цикле, также темно синим красим центральные вершины. В конце выводим на экран начальный граф и граф с циклом.

---

## *Алгоритм выполнения операций на псевдокоде*

---

### *Поиск цикла*

```
ЦИКЛ (j = 0, n-1)
    M[j] = 0
КОНЕЦ_ЦИКЛ
Ks = 0
St[ks] = vn
```

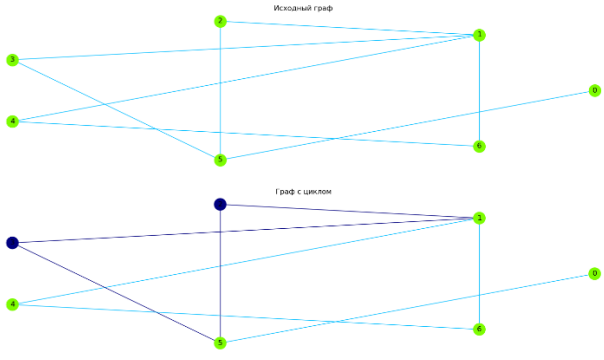
```

M[vn] = 1
L = 0
ЦИКЛ_ПОКА (ks >= 0)
    V = St[ks]
    Pr = 0
    ЦИКЛ (j = L, n)
        ЕСЛИ (A[v, j] = 1) ТО
            ЕСЛИ (j = vk) ТО
                ЕСЛИ (t принадлежит множеству центральных вершин
                не больше двух раз)
                    Вывод (St[i], I = 0, ks), vk
            КОНЕЦ_ЕСЛИ
        ИНАЧЕ
            ЕСЛИ (M[j] = 0) ТО
                Pr = 1
                Прервать цикл по j
            КОНЕЦ_ЕСЛИ
        КОНЕЦ_ЕСЛИ
    КОНЕЦ_ЕСЛИ
    ЕСЛИ (Pr = 1) ТО
        Ks = ks + 1
        St[ks] = j
        L = 0
        M[j] = 1
    ИНАЧЕ
        L = v + 1
        M[v] = 0
        Ks = ks - 1
    КОНЕЦ_ЕСЛИ
КОНЕЦ_ЕСЛИ

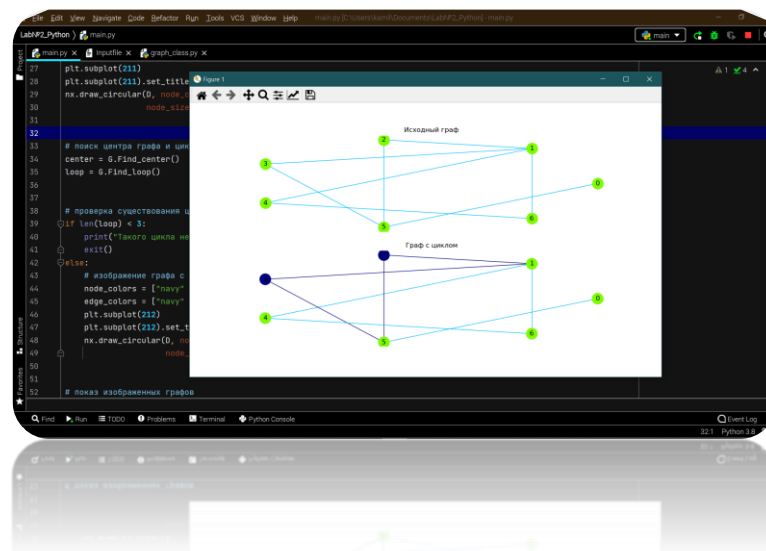
```

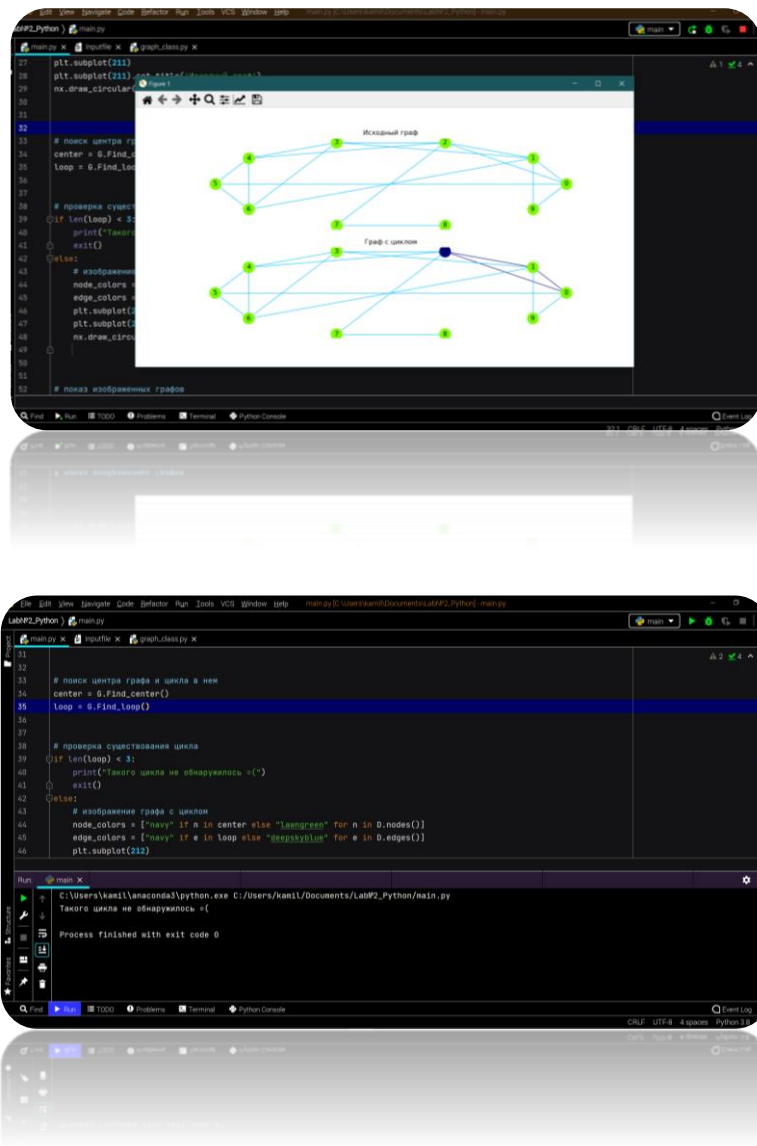
## Тесты

Тест №	Входные данные	Результат
1	0 1 1 0 0 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0	

2	0000010 0011101 0100010 0100010 0100001 1011000 0100100	
3	0001000 0011000 0100101 1100110 0011011 0001101 0010110	Такого цикла не обнаружилось = (

## Распечатки экранов при работе программы





## Листинг программы

### MAIN.py

```
import graph_class
import networkx as nx
import matplotlib.pyplot as plt

# создание области для изображений графов
plt.figure(figsize=(30, 30))

# создание и ввод графа G
G = graph_class.Graph_lab()
f = open('InputFile')
G.Input(f)
f.close()
```

```

# создание графического графа
D = nx.Graph()
for i in range(len(G.A)):
    D.add_node(i)
for i in range(len(G.A)):
    for j in range(len(G.A)):
        if G.A[i][j]:
            D.add_edge(i, j)

# изображаем начальный граф
plt.subplot(211)
plt.subplot(211).set_title('Исходный граф')
nx.draw_circular(D, node_color="lawngreen", with_labels=True,
                 node_size=400, edge_color="deepskyblue")

# поиск центра графа и цикла в нем
center = G.Find_center()
loop = G.Find_loop()

# проверка существования цикла
if len(loop) < 3:
    print("Такого цикла не обнаружилось =(")
    exit()
else:
    # изображение графа с циклом
    node_colors = ["navy" if n in center else "lawngreen" for n in D.nodes()]
    edge_colors = ["navy" if e in loop else "deepskyblue" for e in D.edges()]
    plt.subplot(212)
    plt.subplot(212).set_title('Граф с циклом')
    nx.draw_circular(D, node_color=node_colors, with_labels=True,
                     node_size=400, edge_color=edge_colors)

# показ изображенных графов
plt.show()

print("Конец! Все завершилось успешно!")

```

### Graph\_class.py

```

from math import inf

# класс граф
class Graph_lab(object):
    def __init__(self): # инициализация графа
        self.A = []
        self.size = 0

    def Input(self, f): # ввод графа из файла f
        self.A = [list(map(int, row.split())) for row in f.readlines()]
        self.size = len(self.A)

    def Find_loop(self): # поиск цикла проходящего через не более 2 вершины
        # центра графа
        M = [0] * self.size
        for i in range(self.size):
            M[i] = [0] * self.size

```

```

for i in range(self.size):
    for j in range(self.size):
        if self.A[i][j] == 0:
            M[i][j] = inf
        else:
            M[i][j] = self.A[i][j]
RezMatr = [-1] * 100
for i in range(100):
    RezMatr[i] = [-1] * 100
RezLen = [-1] * 100
St = [-1] * 100
s = list(self.Find_center())
vk = s[0]
vn = s[0]
s.remove(s[0])
center = set(s)
L = 0
kolp = 0
W = [0] * 100
ks = 0
St[ks] = vn
W[vn] = 1
while ks >= 0:
    Pr = 0
    v = St[ks]
    for j in range(L, self.size):
        if M[v][j] == 1:
            M[v][j] = 0
            M[j][v] = 0
            if j == vk:
                Prt = -1
                for i in range(ks + 1):
                    if St[i] in center:
                        Prt += 1
                if Prt <= 0:
                    for i in range(ks + 1):
                        RezMatr[kolp][i] = St[i]
                        RezMatr[kolp][ks + 1] = vk
                        RezLen[kolp] = ks + 2
                        kolp += 1
            else:
                if W[j] == 0:
                    Pr = 1
                    break
    if Pr == 1:
        ks += 1
        St[ks] = j
        L = 0
        W[j] = 1
    else:
        L = v + 1
        W[v] = 0
        ks -= 1
rez = set()
for i in range(len(RezMatr[0])):
    if (RezMatr[0][i] != -1) and (RezMatr[0][i + 1] != -1):
        rez.add((RezMatr[0][i], RezMatr[0][i + 1]))
        rez.add((RezMatr[0][i + 1], RezMatr[0][i]))
return rez

def Find_center(self): # поиск центра графа
    M = [0] * self.size
    for i in range(self.size):
        M[i] = [0] * self.size

```

```

for i in range(self.size):
    for j in range(self.size):
        if self.A[i][j] == 0:
            M[i][j] = inf
        else:
            M[i][j] = self.A[i][j]
e = [0] * self.size
s = []
rad = inf
for i in range(self.size):
    for j in range(self.size):
        for k in range(self.size):
            M[i][j] = min(M[i][j], M[i][k] + M[k][j])
for i in range(self.size):
    for j in range(self.size):
        e[i] = max(e[i], M[i][j])
for i in range(self.size):
    rad = min(rad, e[i])
for i in range(self.size):
    if e[i] == rad:
        s.append(i)
rer = set(s)
return rer

```