|  |  |
| --- | --- |
| |  | | --- | | МИНОБРНАУКИ РОССИИ  Федеральное государственное  бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |

Институт Информационных технологий

Кафедра корпоративных информационных систем

**ЛАБОРАТОРНАЯ РАБОТА**

по дисциплине «Структуры и алгоритмы обработки данных»

**Тема лабораторной работы** Генерация, визуализация и обработка графов.

**Студент группы** ИКБО-12-17 Лисовой А.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Руководитель работы** Андрианова Е.Г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись руководителя)*

Работа представлена к защите «27» апреля 2019 г.

Допущен к защите «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Москва, 2019

**Оглавление**

[Постановка задачи 2](#_Toc7216001)

[Описание алгоритма решения задачи 3](#_Toc7216002)

[Алгоритм: 3](#_Toc7216003)

[Визуализация: 3](#_Toc7216004)

[Тестирование программы 4](#_Toc7216005)

[Задание 1: 4](#_Toc7216006)

[Задание 2: 5](#_Toc7216007)

[Задание 3: 6](#_Toc7216008)

[Задание 4: 7](#_Toc7216009)

[Кодирование алгоритма программы 8](#_Toc7216010)

[Модуль main.py: 8](#_Toc7216011)

[Модуль Interface.py: 10](#_Toc7216012)

[Модуль Graph.py 11](#_Toc7216013)

# Постановка задачи

Сгенерировать случайный социальный граф, описывающий n (> 100) людей, находящихся в отношении "знают друг друга". При этом непосредственных знакомых может быть от 1 до 4. Вывести его на экран.

Задание 1. Для каждого участника найти всех не знакомых с ним людей, имеющих 1 общего друга c данным участником.

Задание 2. Для каждого участника найти всех не знакомых с ним людей, имеющих 2 общих друзей c данным участником.

Задание 3. Для каждого участника найти всех не знакомых с ним людей, имеющих 3 общих друзей c данным участником.

Задание 4. Для каждого участника найти всех не знакомых с ним людей, имеющих 4 общих друзей c данным участником.

# Описание алгоритма решения задачи

## Алгоритм:

1. Случайным образом сгенерировать граф, причем так, чтобы допустимое число ребер от одной вершины к другой в процессе генерации было от 1 до 4 включительно. Граф представляет из себя словари вершин и ребер между ними.
2. Затем, для каждой вершины созданного графа, совершая их обход по порядковому номеру:
3. Найти и составить список дружественных к ней вершин (То есть, всех соседних к ней вершину).
4. Все вершины графа обозначить фиолетовым цветом, текущую вершину графа обозначить красным цветом. Дружественные вершины обозначить оранжевым цветом.
5. После чего пройти по списку всех вершин графа, выделяя подходящие по критерию поиска (1, 2, 3, 4 общих знакомых).
6. Придать всем подходящим по критерию вершинам желтый цвет.
7. Отобразить на изображении сгенерированного графа ситуацию для вершин.
8. Завершить графическое отображение графа, перейти в главное меню программы.

## Визуализация:

Для визуализации графа я воспользовался средствами библиотеки Networkx 2.3, а также совместимым с ней функционалом библиотек PyLab и MatPlotLib.

Граф изображается при помощи силового алгоритма отрисовки графа – Камады-Каваи, основанного на принципе использования сил пружин между всеми парами вершин с идеальными длинами пружин, равными расстоянию по графу, что делает граф более простым для понимания.

# Тестирование программы

## Задание 1:

Тест №1: Для каждого участника найти всех не знакомых с ним людей, имеющих 1 общего друга c данным участником.

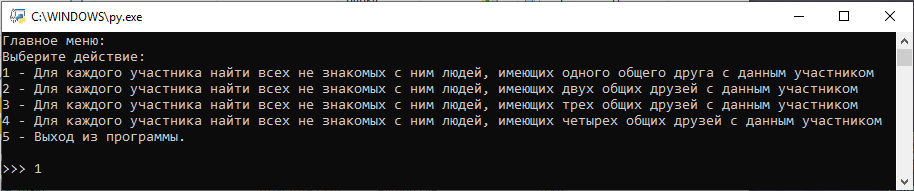


Рис. 1 – Выбор первого задания в главном меню программы.

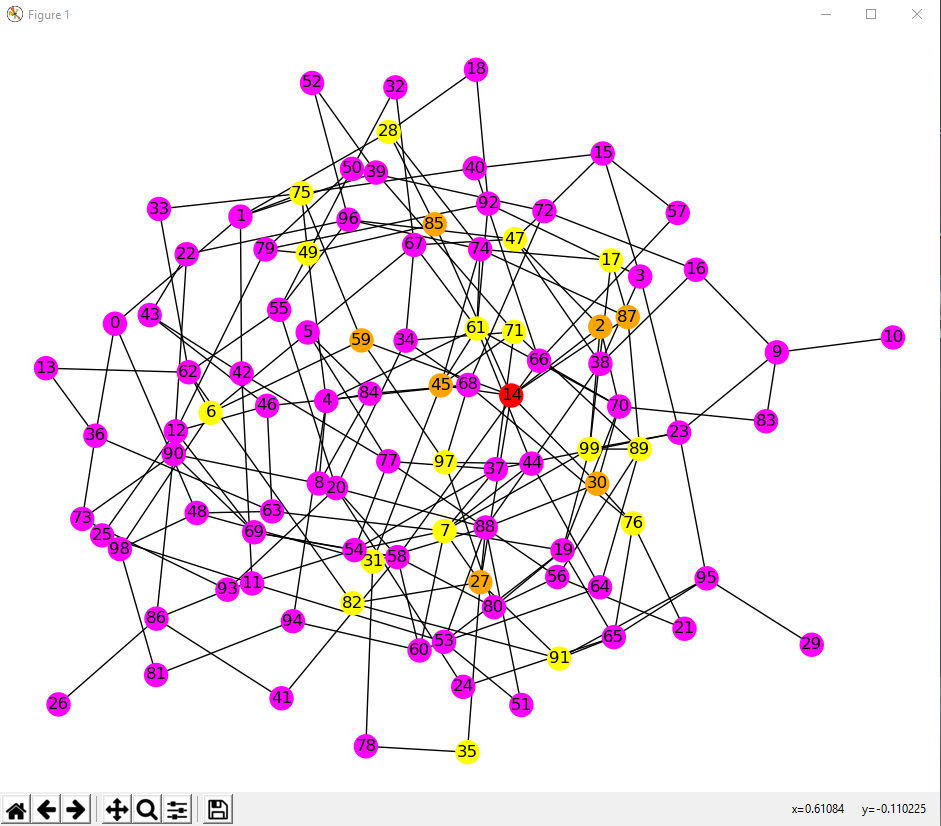


Рис. 2 – Обход графа в ходе выполнения задания 1.

Красным цветом отмечена текущая вершина, желтым – дружественные к ней вершины, желтым – анонимы с одним общим другом с текущей вершиной.

## Задание 2:

Тест №2: Для каждого участника найти всех не знакомых с ним людей, имеющих 2 общих друзей c данным участником.

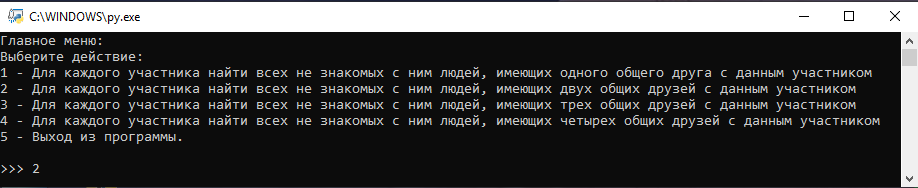


Рис. 3 – Выбор второго задания

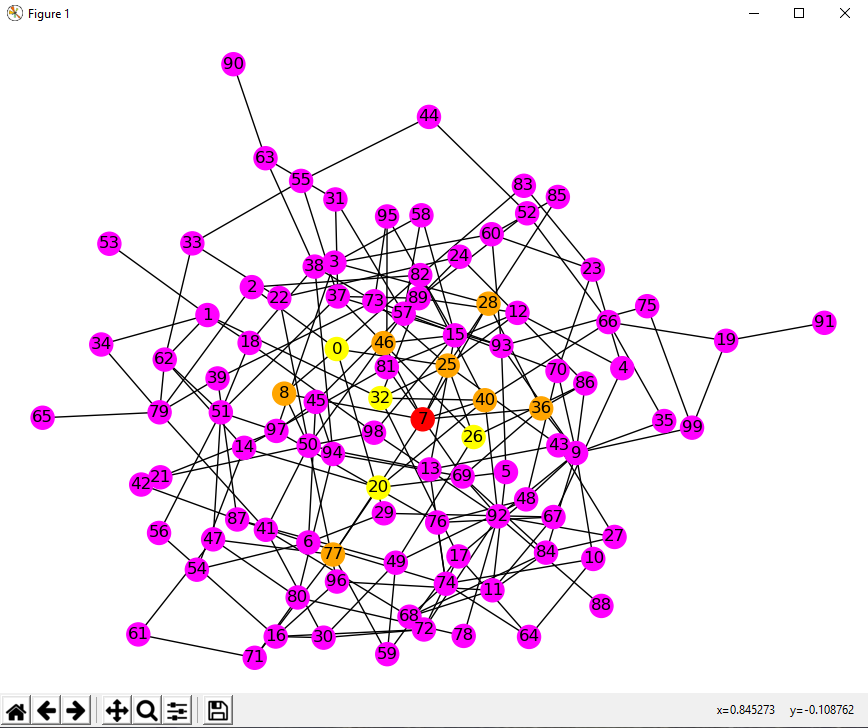


Рис. 4 – Выполнение второго задания.

Необходимо соблюдение условия наличия двух общих друзей между анонимом и текущей вершиной. Видно: У вершины 0 общие друзья 8 и 25; У вершины 32 – общие друзья 25 и 40; У вершины 20 – общие друзья 77 и 40; У вершины 26 – общие друзья 25 и 36. На основании этого можно сделать вывод, что программа работает корректно.

## Задание 3:

Тест №3: Для каждого участника найти всех не знакомых с ним людей, имеющих 3 общих друзей c данным участником.

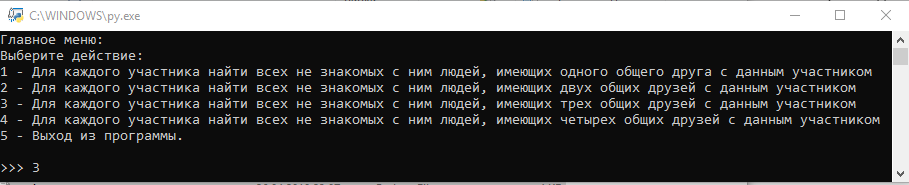


Рис. 5 – Выбор третьего задания.

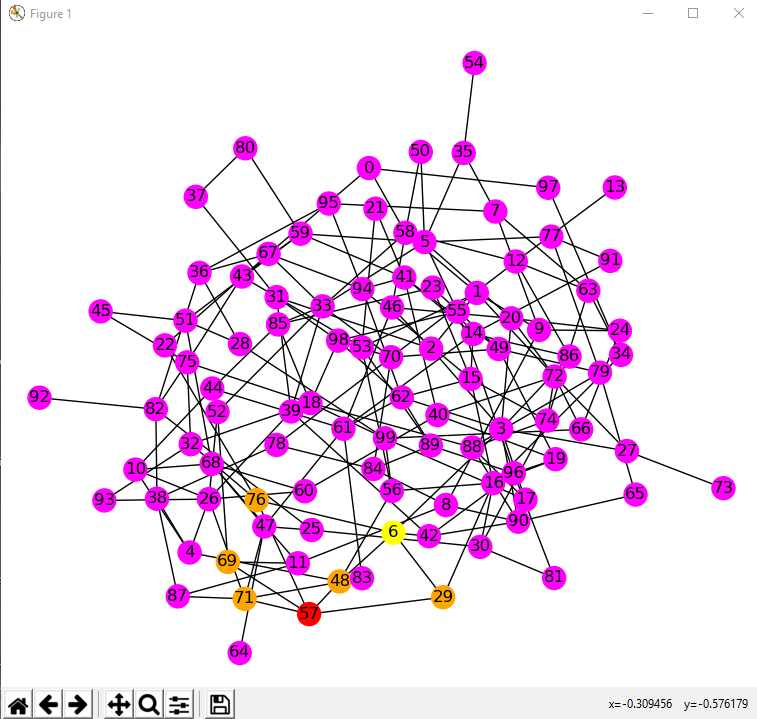


Рис. 6 – Выполнение третьего задания.

Для данной итерации обхода графа справедливо, что у вершины 6 общими друзьями являются вершины 76, 48 и 29. Это соответствует заданию

## Задание 4:

Тест №4: Для каждого участника найти всех не знакомых с ним людей, имеющих 4 общих друзей c данным участником.

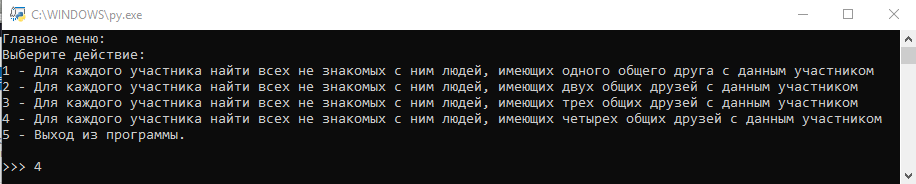


Рис. 7 – Выбор четвертого задания.

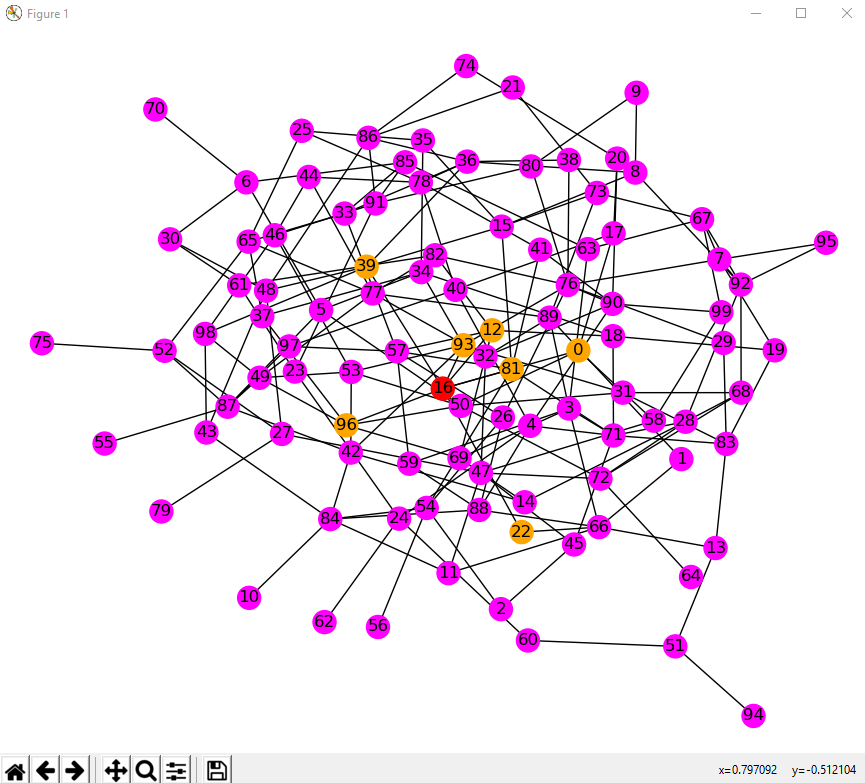


Рис. 8 – Выполнение четвертого задания.

# Кодирование алгоритма программы

## Модуль main.py:

# Список импортов

from Graph import Graph

from Interface import Interface

import networkx as nx

import matplotlib.pyplot as plt

import pylab as pl

# Консольное меню программы

menu = Interface()

# Вызов метода отображения главного меню

menu.interface\_Main()

# Цикл обработки команд

while True:

# Ввод выбранного действия

Step = menu.input\_Controller()

# Задание 1

if Step == 1:

print("")

# Объекта класса графа

task1 = Graph(100, 1)

# Цикл обхода графа и его поэтапной отрисовки

pl.ion()

pl.show()

for vertex in range(task1.user\_value):

pl.clf()

task1.anonymous\_find(vertex)

color\_list = task1.append\_color\_for\_vertex(vertex)

nx.draw\_kamada\_kawai(task1.graph, node\_size = 275, node\_color

= color\_list,

edge\_color = "black", font\_color = "black", font\_weight = 5,

with\_labels = True)

pl.show()

pl.pause(1.0)

menu.interface\_Main()

# Задание 2

elif Step == 2:

print("")

# Объекта класса графа

task2 = Graph(100, 2)

# Цикл обхода графа и его поэтапной отрисовки

pl.ion()

pl.show()

for vertex in range(task2.user\_value):

pl.clf()

task2.anonymous\_find(vertex)

color\_list = task2.append\_color\_for\_vertex(vertex)

nx.draw\_kamada\_kawai(task2.graph, node\_size = 275, node\_color

= color\_list,

edge\_color = "black", font\_color = "black", font\_weight = 5,

backgrond\_color = "beige", with\_labels = True)

pl.show()

pl.pause(1.0)

menu.interface\_Main()

# Задание 3

elif Step == 3:

print("")

# Объекта класса графа

task3 = Graph(100, 3)

# Цикл обхода графа и его поэтапной отрисовки

pl.ion()

pl.show()

for vertex in range(task3.user\_value):

pl.clf()

task3.anonymous\_find(vertex)

color\_list = task3.append\_color\_for\_vertex(vertex)

nx.draw\_kamada\_kawai(task3.graph, node\_size = 275, node\_color

= color\_list,

edge\_color = "black", font\_color = "black", font\_weight = 5,

with\_labels = True, )

pl.show()

pl.pause(1.0)

menu.interface\_Main()

# Задание 4

elif Step == 4:

print("")

# Объекта класса графа

task4 = Graph(100, 4)

# Цикл обхода графа и его поэтапной отрисовки

pl.ion()

pl.show()

for vertex in range(task4.user\_value):

pl.clf()

task4.anonymous\_find(vertex)

color\_list = task4.append\_color\_for\_vertex(vertex)

nx.draw\_kamada\_kawai(task4.graph, node\_size = 275, node\_color

= color\_list,

edge\_color = "black", font\_color = "black", font\_weight = 5,

with\_labels = True)

pl.show()

pl.pause(1.0)

menu.interface\_Main()

# Выход из программы

elif Step == 5:

print("Программа завершена")

break

# Остальные случаи

else:

print("Ошибка ввода")

print("")

## Модуль Interface.py:

# Класс интерфейса

class Interface:

# Конструктор

def \_\_init\_\_(self):

pass

# Главное меню:

def interface\_Main(self):

print ("Главное меню:")

print ("Выберите действие:")

print ("1 - Для каждого участника найти всех не знакомых с ним

людей, имеющих одного общего друга c данным участником")

print ("2 - Для каждого участника найти всех не знакомых с ним

людей, имеющих двух общих друзей c данным участником")

print ("3 - Для каждого участника найти всех не знакомых с ним

людей, имеющих трех общих друзей c данным участником")

print ("4 - Для каждого участника найти всех не знакомых с ним

людей, имеющих четырех общих друзей c данным участником")

print ("5 - Выход из программы.")

print("")

pass

# Метод защиты от некорректного ввода:

def input\_Controller(self):

while True:

try:

Value = int(input(">>> "))

if Value > 0:

return Value

else:

print("Ошибка ввода!")

except ValueError:

print("Ошибка ввода!")

print("")

## Модуль Graph.py

# Список импортов

import random

import networkx as nx

import numpy as np

# Класс графа

class Graph:

# Конструктор

def \_\_init\_\_(self, user\_value, count):

self.user\_value = user\_value

self.anonymous\_legion = []

self.count = count

self.graph = nx.Graph()

# Цикл добавления вершин в социальный граф

for vertex in range(self.user\_value):

self.graph.add\_node(vertex)

# Цикл создания связей между вершинами

for vertex in range(self.user\_value):

neighbors = len([self.graph.neighbors(vertex)])

if neighbors < 4:

edges = random.randint(1, 4 - neighbors)

# Цикл создания связей

while edges > 0:

neighbor = random.randint(0, self.user\_value - 1)

if neighbor != vertex or

len(list(self.graph.neighbors(neighbor))) <= 3:

self.graph.add\_edge(vertex, neighbor)

edges = edges - 1

# Метод поиска незнакомцев с определенным количеством общих знакомых

def anonymous\_find(self, current\_user):

friendly\_users = list(self.graph.neighbors(current\_user))

# Цикл обхода всех вершин графа

for vertex in range(self.user\_value):

if vertex != current\_user and vertex not in friendly\_users:

number = 0

current\_friendly\_list = list(self.graph.neighbors(vertex))

# Цикл обхода ребер графа (друзей пользователя)

for jertex in friendly\_users:

if jertex in current\_friendly\_list:

number = number + 1

if number == self.count:

self.anonymous\_legion.append(vertex)

# Метод придания искомым вершинам цветовой окраски

def append\_color\_for\_vertex(self, current\_user):

friendly\_users = list(self.graph.neighbors(current\_user))

colour\_list = []

# Цикл придания цвета

for index in range(self.user\_value):

colour\_list.append("magenta")

index = index + 1

# Цикл разукрашивания анонимусов

for anonymous in self.anonymous\_legion:

colour\_list[anonymous] = "yellow"

# Цикл разукрашивания друзей

for friendly\_user in friendly\_users:

colour\_list[friendly\_user] = "orange"

colour\_list[current\_user] = "red"

self.anonymous\_legion = []

return colour\_list