**Main.py**

from Form import \*  
  
# окно приложения  
window = Tk()  
window.geometry("1200x900")  
window.title("Сиаод, 3 лабораторная, Литвненко Борис ИКБО-06-18")  
g = Graph() # граф с которым производятся действия  
w = Form(window, g) # создаю форму и указываю с каким графом надо работать  
window.mainloop()

**From.py**

# импортируем необходимые библиотеки  
from tkinter import \*  
import tkinter.ttk as ttk  
from tkinter import messagebox  
from Graph import \*  
  
  
class Form():  
 def \_\_init\_\_(self, root, graph): # root - форма размещения, graph - граф для работы  
 self.canvas = \  
 Canvas(root, width=WIDTH, height=HEIGHT, background="White") # "холст" рисования графа  
  
 self.frame = Frame(root) # рамка для размещения объектов добавления данных  
 self.buttonNew = Button(self.frame, text="Создать", command=self.initialize) # кнопка добавления количества вершин графа  
 self.buttonFile = Button(self.frame, text="Взять из матрицы", command=self.initializeFile) # по нажатии берется матрица из файла мatrix.txt  
 self.button2 = Button(self.frame, text="Добавить", command=self.add\_edge) # кнопка добавления связи между графами  
 self.buttonCount = Button(root, text="Нати наибольший простой путь (пути)", command=self.count) # Начать задание  
 self.label = Label(self.frame, text="Количество вершин будущего графа") # информационный label  
 self.label.pack(side=LEFT) # показываем информационный label  
 self.nEntry = StringVar() # для чтения из текстового поля  
 self.lineEdit = Entry(self.frame, textvariable=self.nEntry) # поле для ввода  
 self.lineEdit.pack(side=LEFT, padx=10, pady=10) # размещаем поле и задаем пространство между виджетами  
 self.buttonNew.pack(side=LEFT, padx=10, pady=10) # размещаем кнопку и задаем пространство между виджетами  
 self.buttonFile.pack(side=LEFT, padx=10, pady=10) # размещаем кнопку и задаем пространство между виджетами  
  
 self.frame.pack()  
 self.labelResult = Label(root, text="") # поле вывода ответа  
 self.canvas.pack(side=BOTTOM) # добавляем на рамку поле для рисования  
 self.buttonCount.pack(side=BOTTOM)  
 self.labelResult.pack(side=BOTTOM)  
 self.graph = graph  
  
 def initialize(self):  
 try:  
 self.n = int(self.lineEdit.get())  
 if self.n <= 0:  
 raise ValueError  
 self.label["text"] = "Добавить ребро (пример: 1-2)"  
 self.graph.setSize(self.n) # устанавливаем размер графа  
 self.buttonNew.pack\_forget() # убираем кнопку  
 self.buttonFile.pack\_forget() # и эту  
 self.button2.pack(side=LEFT)  
 self.lineEdit.delete(0, END) # очищаем поле ввода  
 self.draw()  
 except ValueError:  
 messagebox.showinfo('Ошибка', 'Проверьте правильность заполнения поля')  
  
 def initializeFile(self):  
 try:  
 # в файле граф представлен в виде матрицы смежности  
 with open('matrix.txt', 'r') as f:  
 matrix = f.read().splitlines()  
 self.len = len(matrix) # сторона матрицы  
 self.graph.setSize(self.len) # устанавливаем размер графа  
 # добавляем все связи  
 for n\_line in range(self.len):  
 for n\_char in range(self.len):  
 if matrix[n\_line][n\_char] == "1":  
 self.graph.addEdge(n\_line+1, n\_char+1)  
 self.draw() # отрисовываем граф  
 self.buttonNew.pack\_forget() # убираем кнопку  
 self.buttonFile.pack\_forget() # и эту  
 self.button2.pack(side=LEFT) # и ставим нужную, что б можно было дополнить граф  
 self.label["text"] = "Добавить ребро (пример: 1-2)"  
 except FileNotFoundError:  
 messagebox.showinfo('Ошибка', 'Ошибка открытия файла')  
 except Exception:  
 messagebox.showinfo('Ошибка', 'Ошибка при заполнении')  
  
 def add\_edge(self):  
 line = self.lineEdit.get()  
 self.n = self.graph.size  
 try:  
 line.split(",")  
 for l in line:  
 a = int(line.split('-')[0]) # разбиваем введенную строку на два числа  
 b = int(line.split('-')[1])  
 self.lineEdit.delete(0, END) # очищаем поле ввода  
 if (a < 1 or b < 1 or a > self.n or b > self.n or a == b): # если номер ребра неправильный  
 messagebox.showinfo('Ошибка', 'Некорректные значения')  
 return  
 self.graph.addEdge(a, b) # добавляем новую связь  
 self.draw() # рисуем заново граф со связями  
 except Exception:  
 messagebox.showinfo('Ошибка', 'Проверьте правильность заполнения полей - две цифры через тире (1-2)')  
  
 # красиво записывает список в строку [1, 2, 3] -> "1 2 3"  
 def str\_p(self, path):  
 res = ""  
 for n in path:  
 res += (str(n) + " ")  
 return res  
  
 # выполняет задание  
 def count(self):  
 self.graph.paths = [] # обнуляем все пути на случай, если задание выполняется повторно  
 # перебирает все возможные сочетания вершин "от до",  
 # находит все возможные простые пути между каждо парой вершин и  
 # и записывает в paths  
 for i in range(len(self.graph.nodes)):  
 for j in range(len(self.graph.nodes)):  
 if i == j:  
 continue  
 self.graph.findPaths(i + 1, j + 1)  
 # вывод ответа на задание  
 l\_max = 0 # максимальная длина  
 result = "" # итоговый результат  
 for path in self.graph.paths:  
 if len(path) > l\_max:  
 l\_max = len(path)  
 result = self.str\_p(path)  
 elif len(path) == l\_max:  
 result += ('\n' + self.str\_p(path))  
 result = "Ответ: максимальная длина - " + str(l\_max) + ", у простого путя (путей): \n" + result  
 self.labelResult["text"] = result  
  
 """следующие методы для открисовки графа"""  
 def draw\_node(self, x, y, text, r=RADIUS):  
 self.canvas.create\_oval(x - r, y - r, x + r, y + r, fill="MistyRose") # рисуем узлы  
 self.canvas.create\_text(x, y, text=text) # добавляем подписи  
  
 def draw\_graph(self):  
 for s in self.graph.nodes: # проходим по списку узлов  
 for t in s.targets: # проходим по списку его соседей  
 self.canvas.create\_line(s.x, s.y, t.x, t.y, width=2, arrow=LAST,  
 arrowshape="15 50 15") # отрисовываем линию соединяющую вершины  
 for n in self.graph.nodes:  
 self.draw\_node(n.x, n.y, n.label) # отрисовываем узел  
  
 def draw(self):  
 self.canvas.delete("all") # очищаем рисунок  
 self.draw\_graph() # рисуем заново

**Graph.py**

import random  
  
"""глобальные переменные"""  
RADIUS = 15 # радиус узлов  
WIDTH, HEIGHT = 900, 600 # размеры поля  
  
  
# узел графа  
class Node:  
 def \_\_init\_\_(self, label, x, y):  
 self.label = label # имя  
 self.targets = [] # спиисок связей  
 # координаты вершины  
 self.x = x  
 self.y = y  
  
 def connect(self, node): # соединение узлов  
 if node not in self.targets: # если такая связи не было  
 self.targets.append(node) # добавляем новую связь  
  
  
class Graph: # класс для представления графа  
 def \_\_init\_\_(self):  
 self.nodes = [] # список узлов  
 self.size = 0 # количество вершин  
 self.paths = [] # список простых путей  
  
 def setSize(self, n): # задает количество узлов графа и определет координаты каждого узла  
 self.size = n  
 # генерация узлов (присвоение имени и координат)  
 for i in range(1, (n + 1)):  
 x = random.randint(RADIUS \* 4, WIDTH - RADIUS \* 4 - 1) # получаем случайные значения для координат  
 y = random.randint(RADIUS \* 4, HEIGHT - RADIUS \* 4 - 1)  
 new\_node = Node(i, x, y) # создаем узел графа  
 self.nodes.append(new\_node) # добавляем в список узлов  
  
 def addEdge(self, a, b): # созать новую связь между узлами  
 ind = a - 1 # от этого узла связь  
 ind2 = b - 1 # связь к этому узлу  
 self.nodes[ind].connect(self.nodes[ind2]) # добавляем связь  
  
 # находит все возможные простые пути между двумя вершинами графа  
 # рекурсивно. Затем записывает в paths  
 def findPaths(self, a, b, path=None):  
 if path == None:  
 path = [a]  
 for node in self.nodes[a - 1].targets: # проходим по связям узла а  
 m\_path = [i for i in path]  
 if node.label != b and node.label not in path:  
 m\_path.append(node.label)  
 self.findPaths(node.label, b, m\_path)  
 elif node.label == b:  
 m\_path.append(node.label)  
 self.paths.append(m\_path)