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

Институт информационных технологий

Кафедра корпоративных информационных систем

|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ**  **по лабораторной работе №3** | | |
| **по дисциплине** | | |
| **«Структуры и алгоритмы обработки данных»**  **Тема лабораторной работы: «**Графовые структуры данных**»** | | |
| Студент группы | ИКБО-06-18 | Литвиненко Б.В. |
| Принял | ассистент кафедры КИС | Исаева И.А. |
|  |  |  |
| Выполнено | «\_\_\_» \_\_\_\_\_\_ 20\_\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись студента)* |
| Зачтено | «\_\_\_» \_\_\_\_\_\_ 20\_\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись преподавателя)* |

Москва 2020

**1. Постановка задачи**

Найти самый длинный простой путь в графе.

**2. Изучение предметной области**

Граф — это совокупность объектов со связями между ними.

**Классификация графов**

Графы делятся на

* связные
* несвязные

В связном графе между любой парой вершин существует как минимум один путь.

В несвязном графе существует хотя бы одна вершина, не связанная с другими.

Графы также подразделяются на

* ориентированные
* неориентированные
* смешанные

В ориентированном графе ребра являются направленными, т.е. существует только одно доступное направление между двумя связными вершинами.

В неориентированном графе по каждому из ребер можно осуществлять переход в обоих направлениях.

Частный случай двух этих видов – смешанный граф. Он характерен наличием как ориентированных, так и неориентированных ребер.

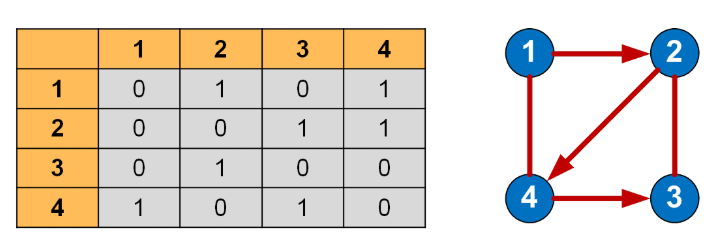
**Способы представления графа**

Граф может быть представлен (сохранен) несколькими способами:

* матрица смежности;
* матрица инцидентности;
* список смежности (инцидентности);
* список ребер.

Использование двух первых методов предполагает хранение графа в виде двумерного массива (матрицы). Размер массива зависит от количества вершин и/или ребер в конкретном графе.

Матрица смежности графа — это квадратная матрица, в которой каждый элемент принимает одно из двух значений: 0 или 1.

  
Рис.1 – «Пример матрицы смежности графа»

**3. Описание используемых структур данных**

Можно было использовать готовые решения (например networkX, graphics или matplotlib) для создания структуры графа и его визуализации. Однако, в целях обучения я принял решение реализовать собственную структуру графа, а так же собственные методы его визуализации.

В файле Graph.py я описал структура графа и методы работы с ним.

Самые основные:

# 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

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

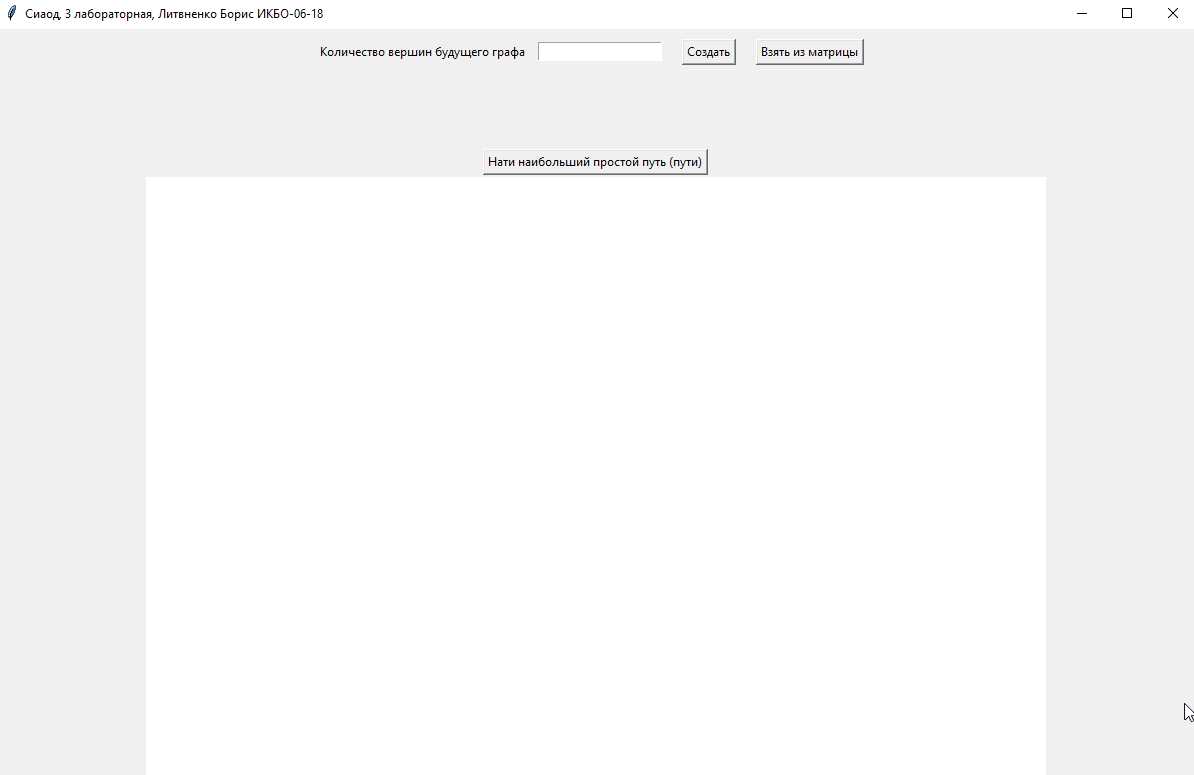
def addEdge(self, a, b): # созать новую связь между узлами

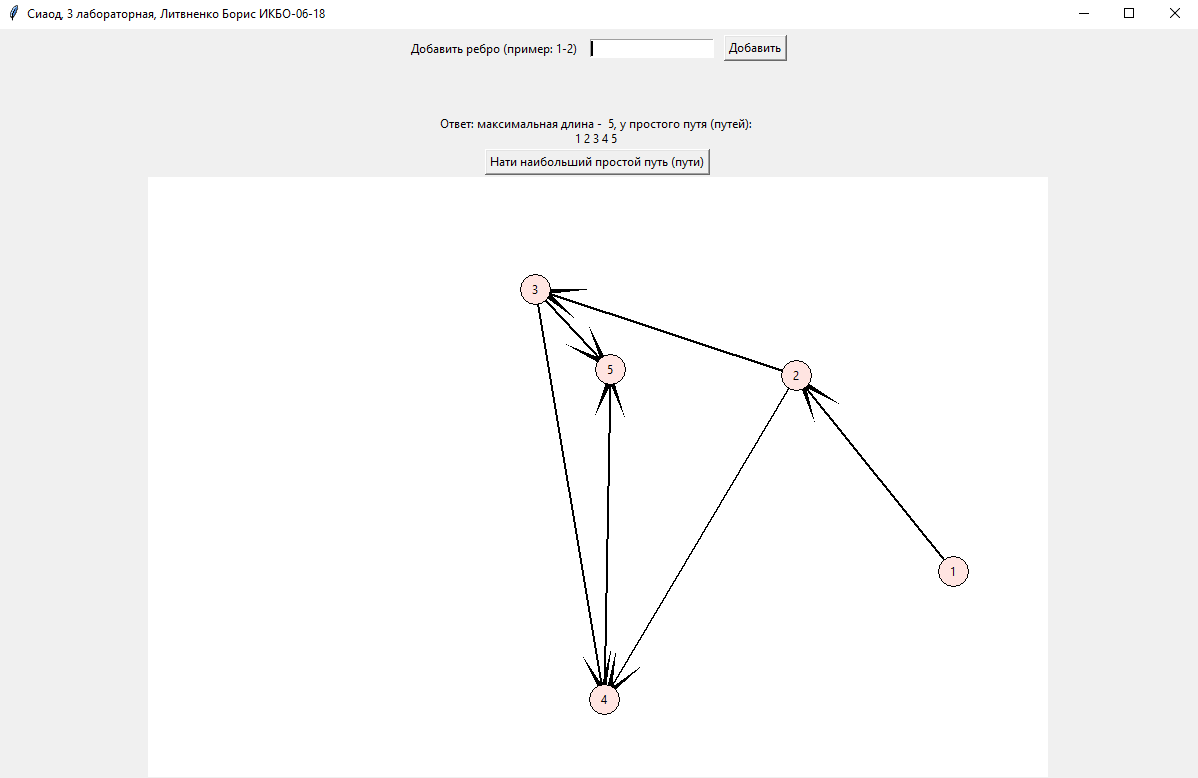
# находит все возможные простые пути между двумя вершинами графа  
# рекурсивно. Затем записывает в paths  
def findPaths(self, a, b, path=None):

(Полный код представлен в Приложении 1 «Листинг проекта»)

**4. Пользовательский интерфейс**

Интерфейс предельно понятен. Существует два способа заполнения графа: создание нового (на первом шаге указывается количество вершин, затем добавляются связи (ребра) по одной в ручном режиме), или создать граф из матрицы смежности, которая записана в файле matrix.txt, который лежит в папке проекта. Так же реализована возможность к этому графу в ручном режиме добавлять связи (ребра).

  
Рис.2 – «Интерфейс 1»

  
Рис.3 – «Интерфейс 2»

**5. Описание алгоритма**

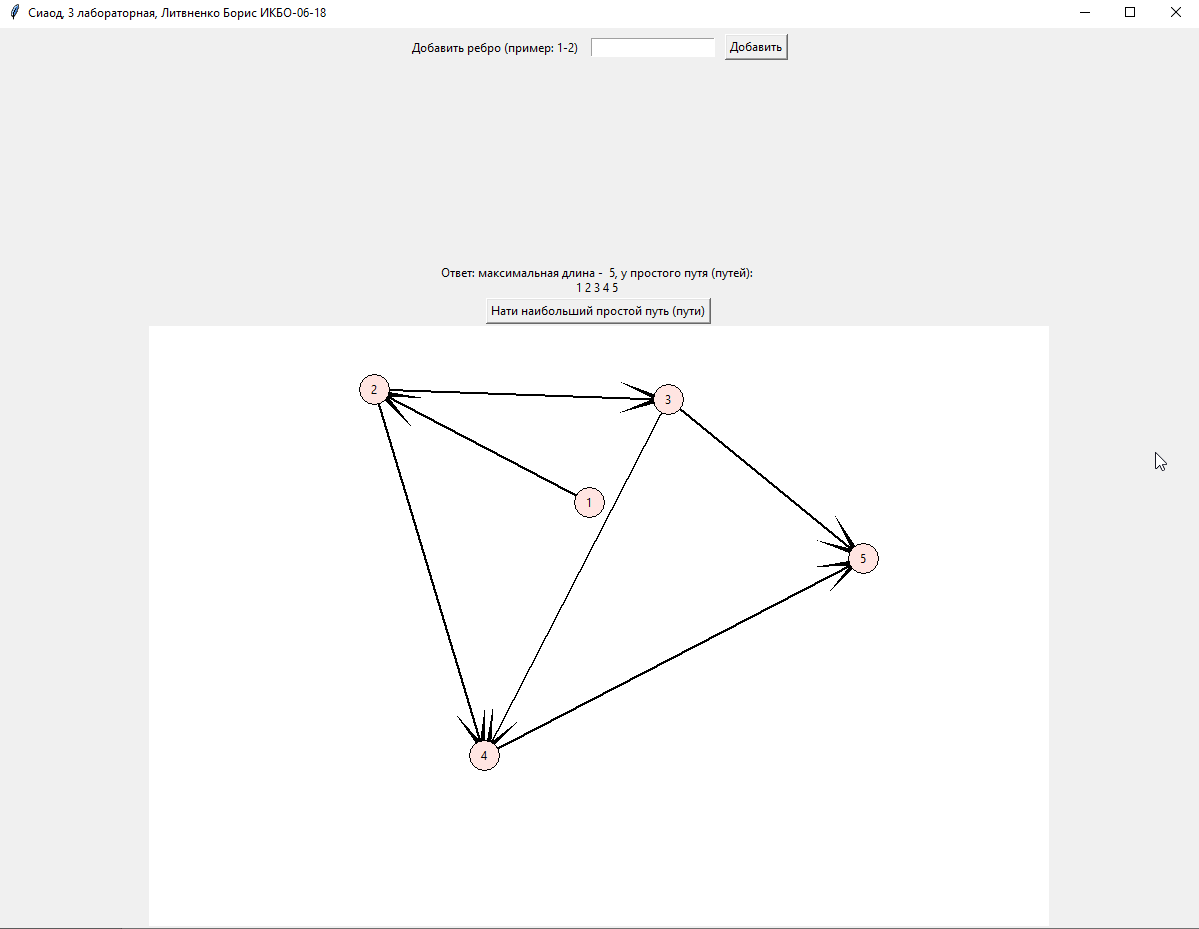
Задача о самом длинном пути — это задача поиска простого пути максимальной длины в заданном графе. Для случаев ориентированных ациклических графов имеет решение за линейное время. В общем же случае это NP-трудная задача, и решается она полным перебором всех возможных путей графа и последующим поиском одного или нескольких самых длинных путей. Именно такой алгоритм организован в моей лабораторной работе.

После заполнения данных, граф готов к работе. По нажатии кнопки «Найти наибольший простой путь (пути)» вызывается функция count модуля Form. Эта функция перебирает все возможные пары вершин графа и с помощью метода findPaths формы Graph записывает в список Paths все возможные простые пути в графе. Затем ищет среди них один самый длинный (или несколько одинаковых по длине самых длинных) путь и выводит результат.

Метод findPaths находит все возможные пути между двумя переданными ему вершинами графа. Поиск реализован самым банальным образом: рекурсией.

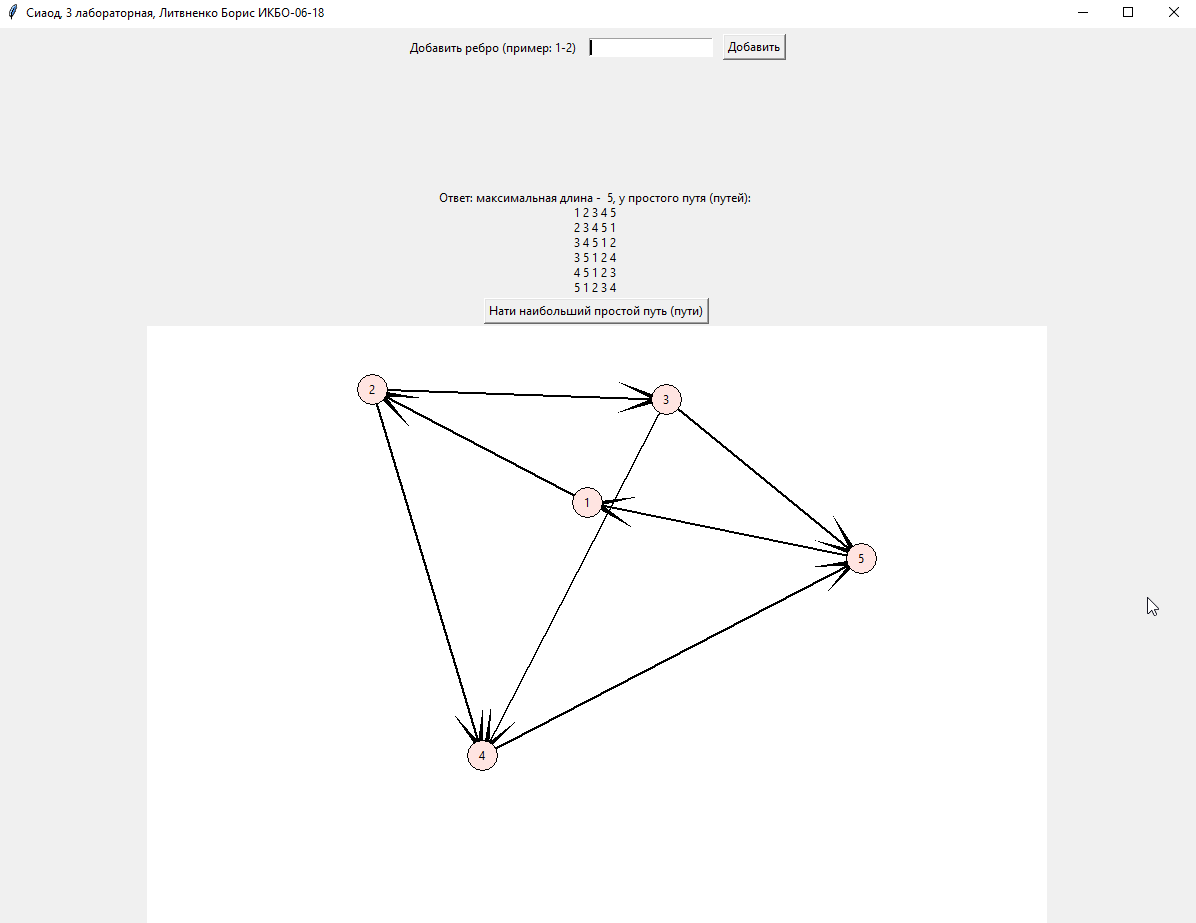
**6. Тестирование программы**

Первое заполнение проведу из матрицы смежности файла matrix.txt.

Рис.4 – «Тестирование»

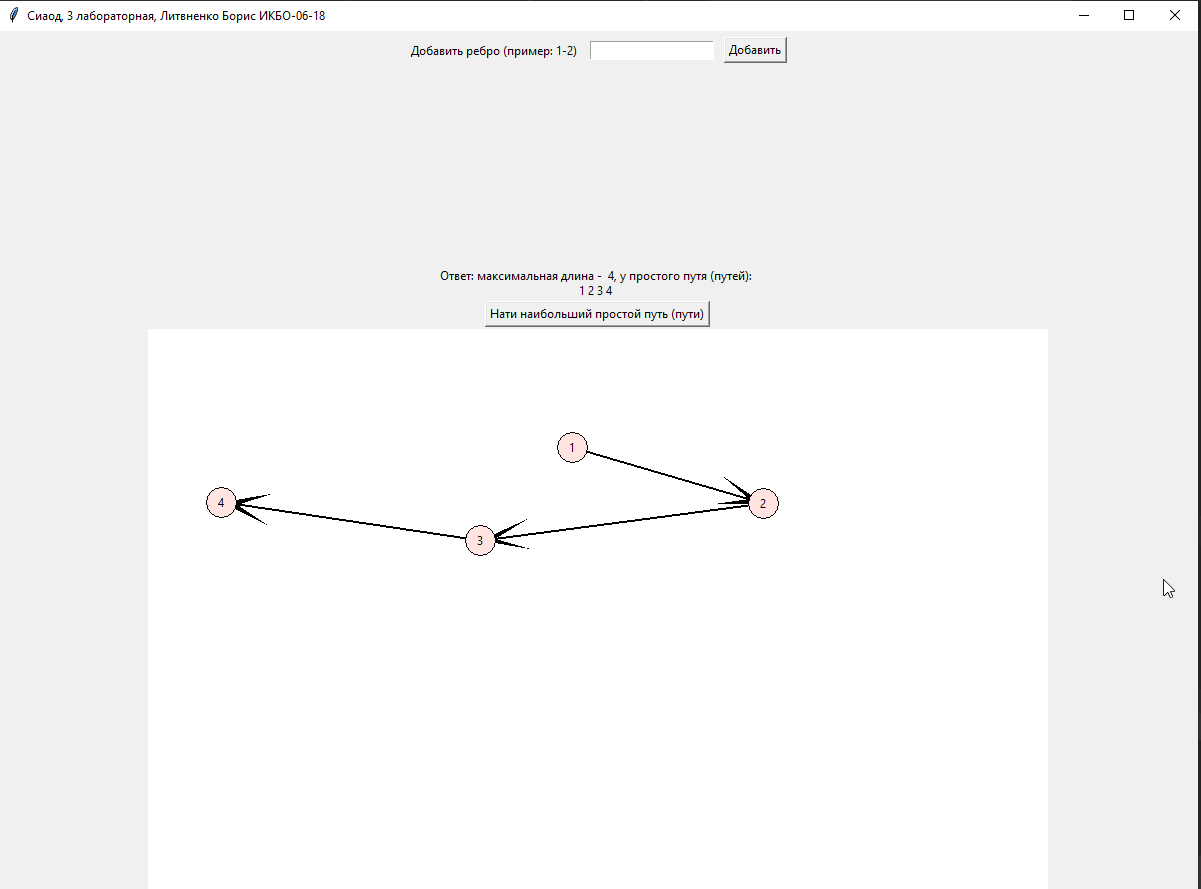
Заполнение прошло без ошибок, так же программа выдает правильный ответ.

А теперь попробую в этот же граф добавить ребро 5-1.

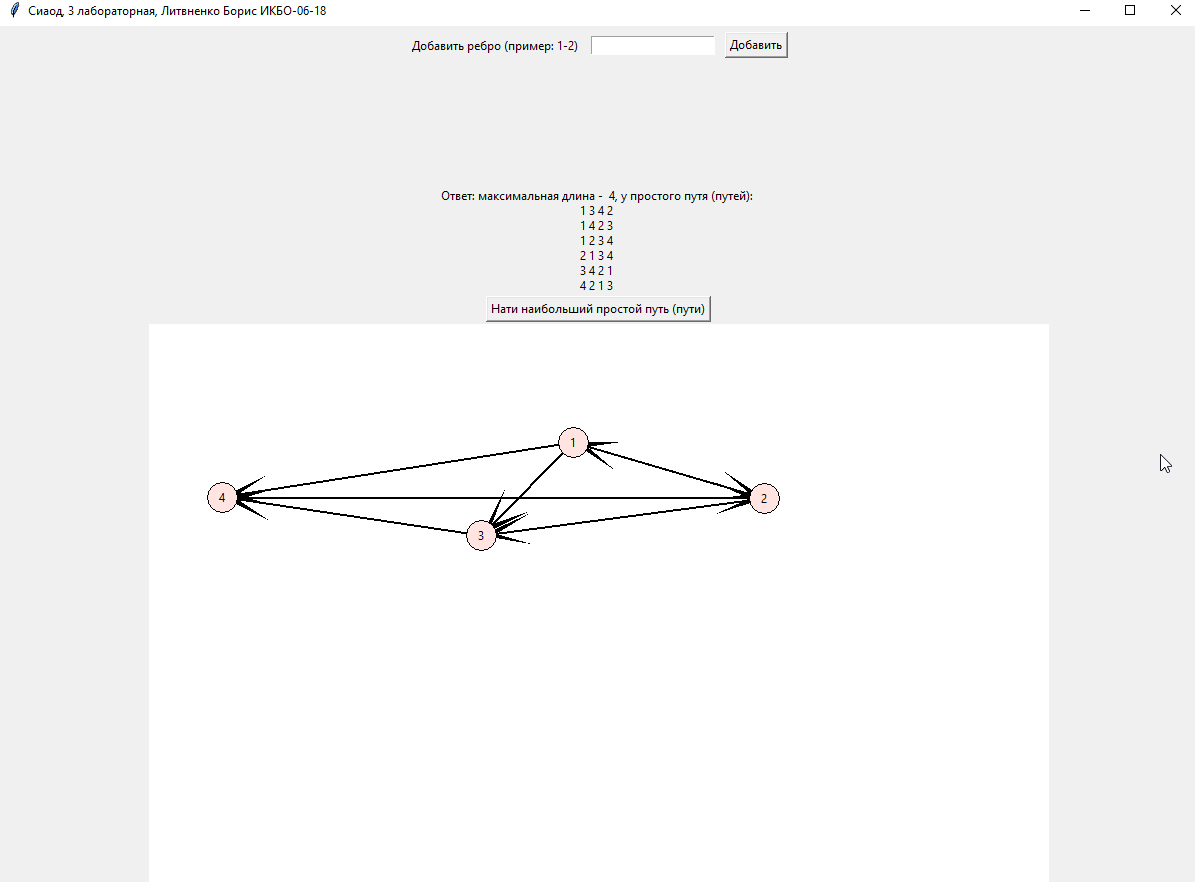
  
Рис.5 – «Тестирование»

Как видно на скриншоте, ребро успешно добавилось. После повторной проверки найдено уже 6 простых самых длинных путей одинаковой длины.

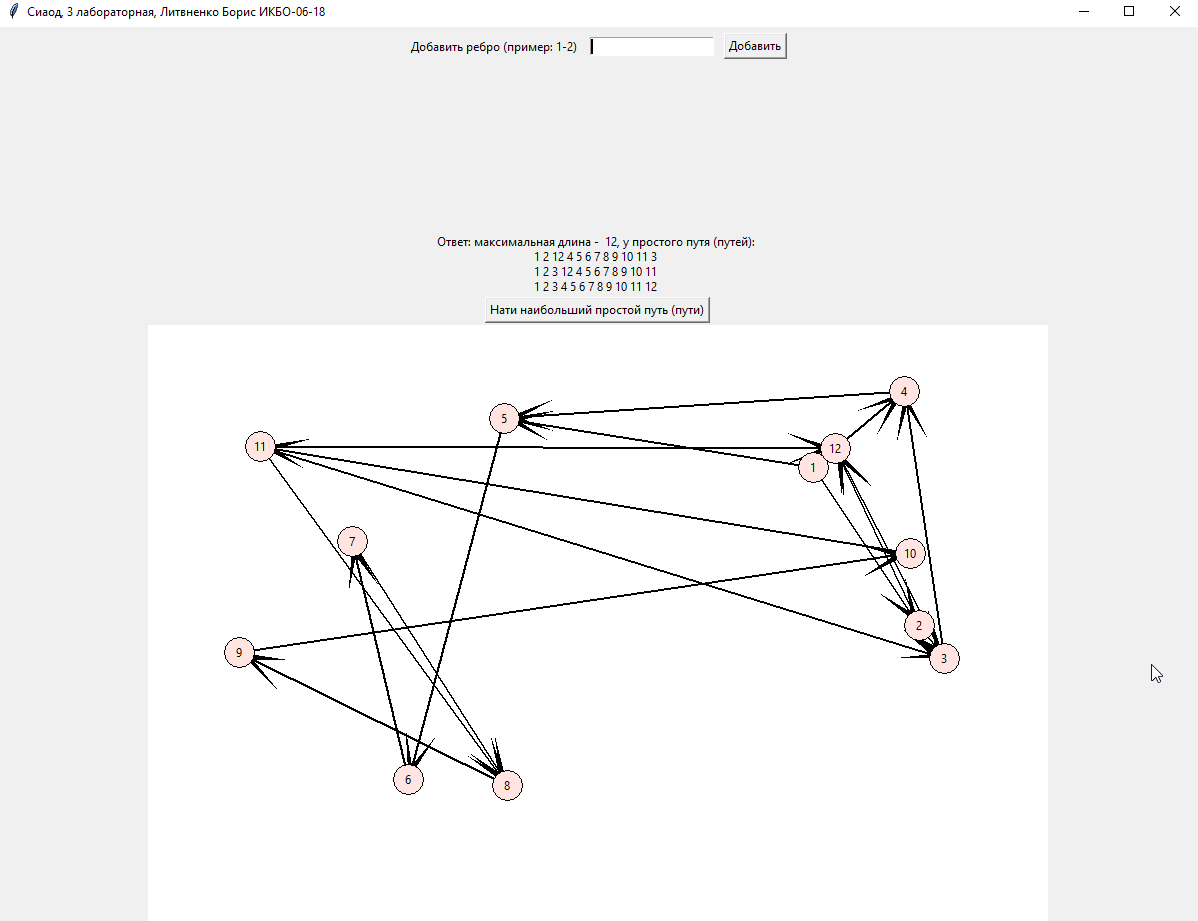
Далее в ручном режиме введу граф из четырех последовательно соединенных узлов.

  
Рис.6 – «Тестирование»

А теперь добавлю в этот же граф несколько связей.

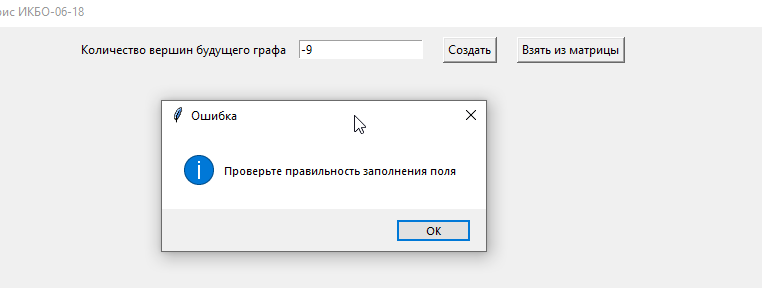
  
Рис.7 – «Тестирование»

А теперь создам граф с 12 вершинами и некоторым количеством связей между ними.

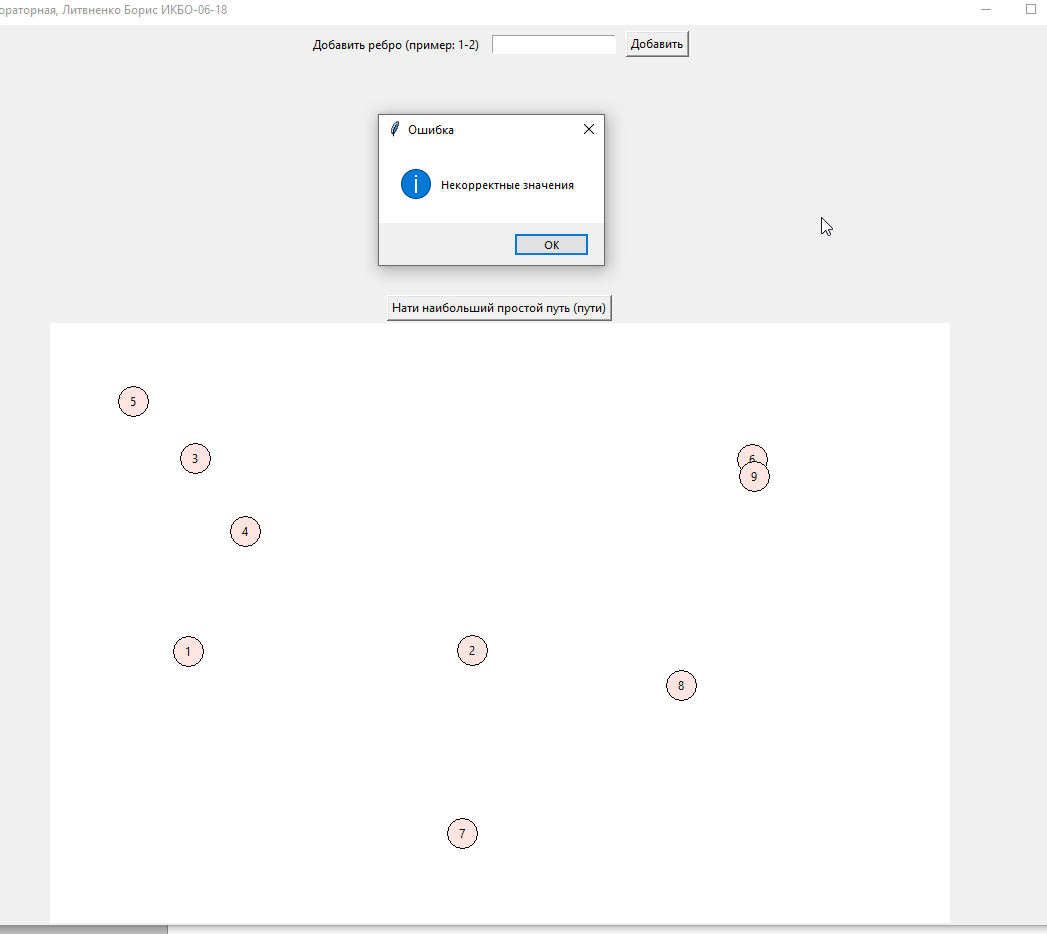
  
Рис.8 – «Тестирование»

Затем проведу тестирование на «устойчивость» программы.

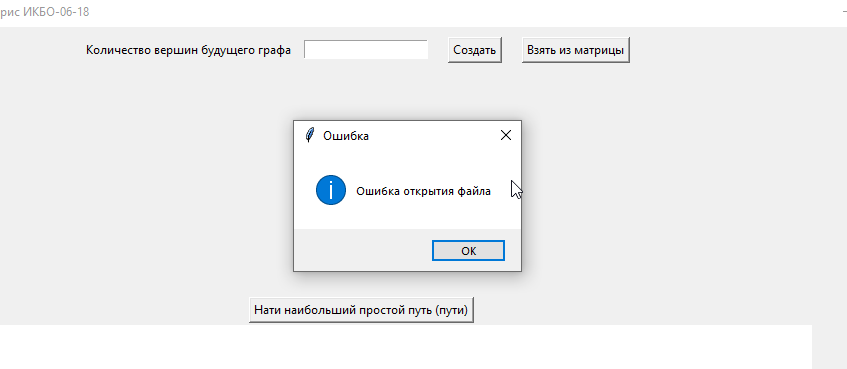
При вводе отрицательного количества числа вершин графа получаем

  
Рис.9 – «Тестирование»

При вводе связи с несуществующей вершиной

  
Рис.10 – «Тестирование»

Если не будет найден файл с матрицей смежности при заполнении от этого файла

Рис.11 – «Тестирование»

В результате тестирования ошибок не выявлено. Программа всегда правильно выполняет подсчет, ошибок в терминале не возникает.

**7. Листинг программы**

В данном пункте приведу код основных методов. Полный листинг в приложении.

# 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

class Graph: # класс для представления графа

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

self.nodes = [] # список узлов

self.size = 0 # количество вершин

self.paths = [] # список простых путей

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)