|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**Лабораторная работа №5**

**«Модели вычислительных алгоритмов»**

**ДИСЦИПЛИНА: «Моделирование»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-72Б | |  |  | ( | Сафронов Н.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Никитенко У.В. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2023

**Цель работы:** изучение технологии математического моделирования вычислительных алгоритмов, моделирование вычислительного алгоритма для оценки его трудоемкости, реализация математической модели на ЭВМ.

**Постановка задачи**

1. Построить по таблице 1, в соответствии с вариантом задания, граф алгоритма.
2. Построить математическую модель вычислительного процесса для оценки трудоемкости алгоритма по методу теории марковских цепей.
3. Построить математическую модель вычислительного процесса для оценки трудоемкости алгоритма сетевым методом.
4. Подготовить программу для расчета модельных характеристик трудоемкости на одном из языков высокого уровня.
5. Подготовить в объектно-ориентированной среде разработки интерактивную форму для управления работой программы и визуализации полученных результатов.

**Вариант 3**

**Таблица 1**

Вариант графа алгоритма

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант 3 |  |  |  |  |  |  |  |  |  |  |  |
| 1.0 |  |  | 0.1 | 0.3 | 0.6 |  |  | 1.0 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| 1.0 |  | 1.0 |  | 0.9 | 0.1 |  |  |  |  |  |

**Таблица 2**

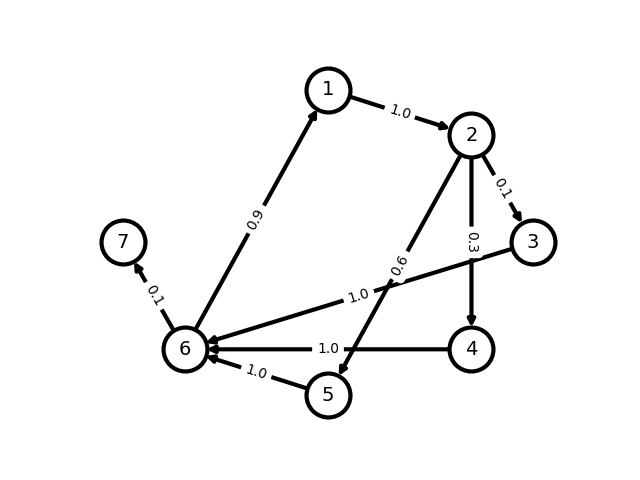
Тип и трудоёмкость операторов

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант 3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 120 | 120 | 150 | 200 | 300 | 600 | 300 | - |

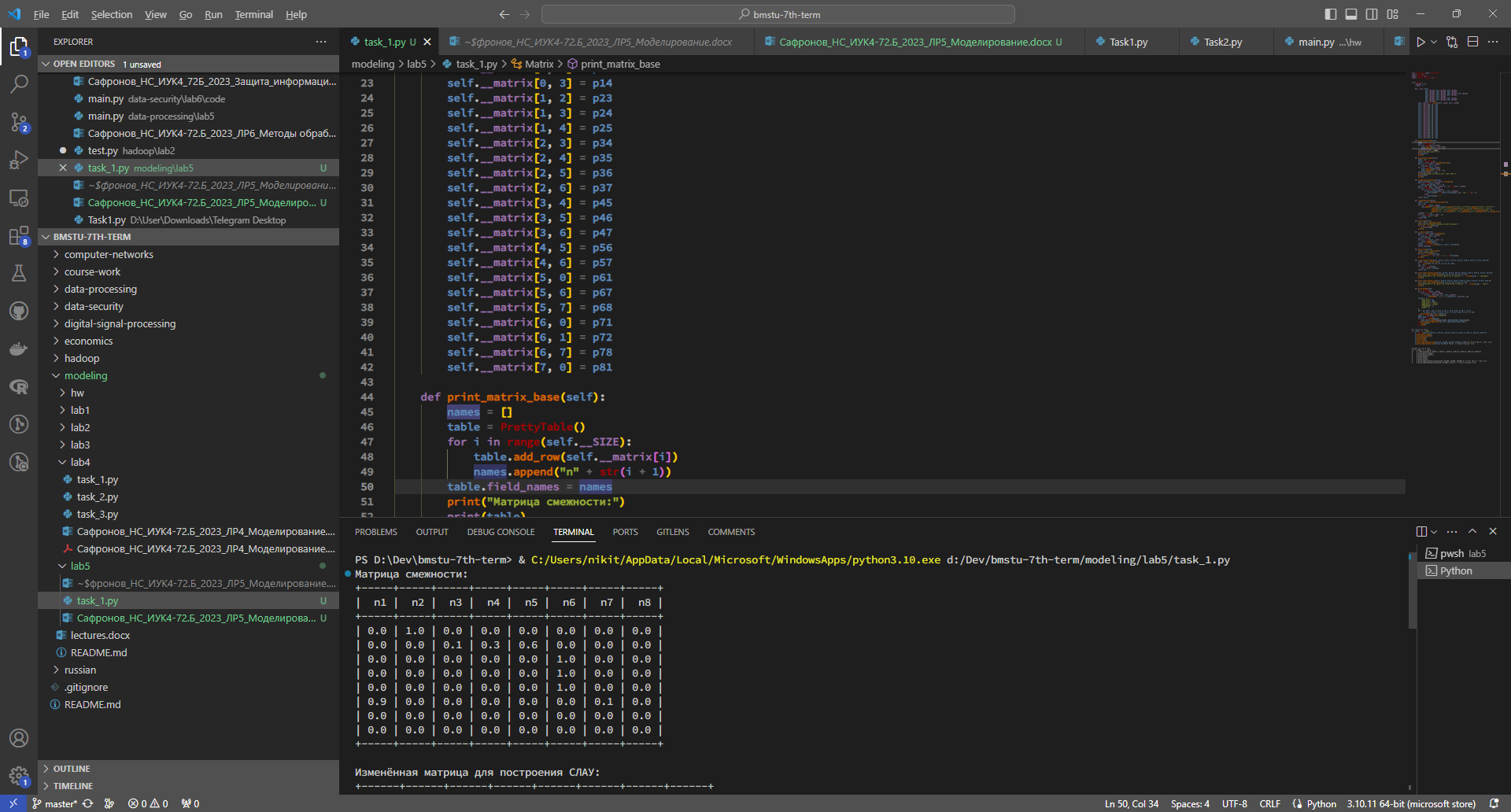
где затемнённые ячейки – ячейки, которые являются операторами ввода/вывода.

**Ход выполнения работы**

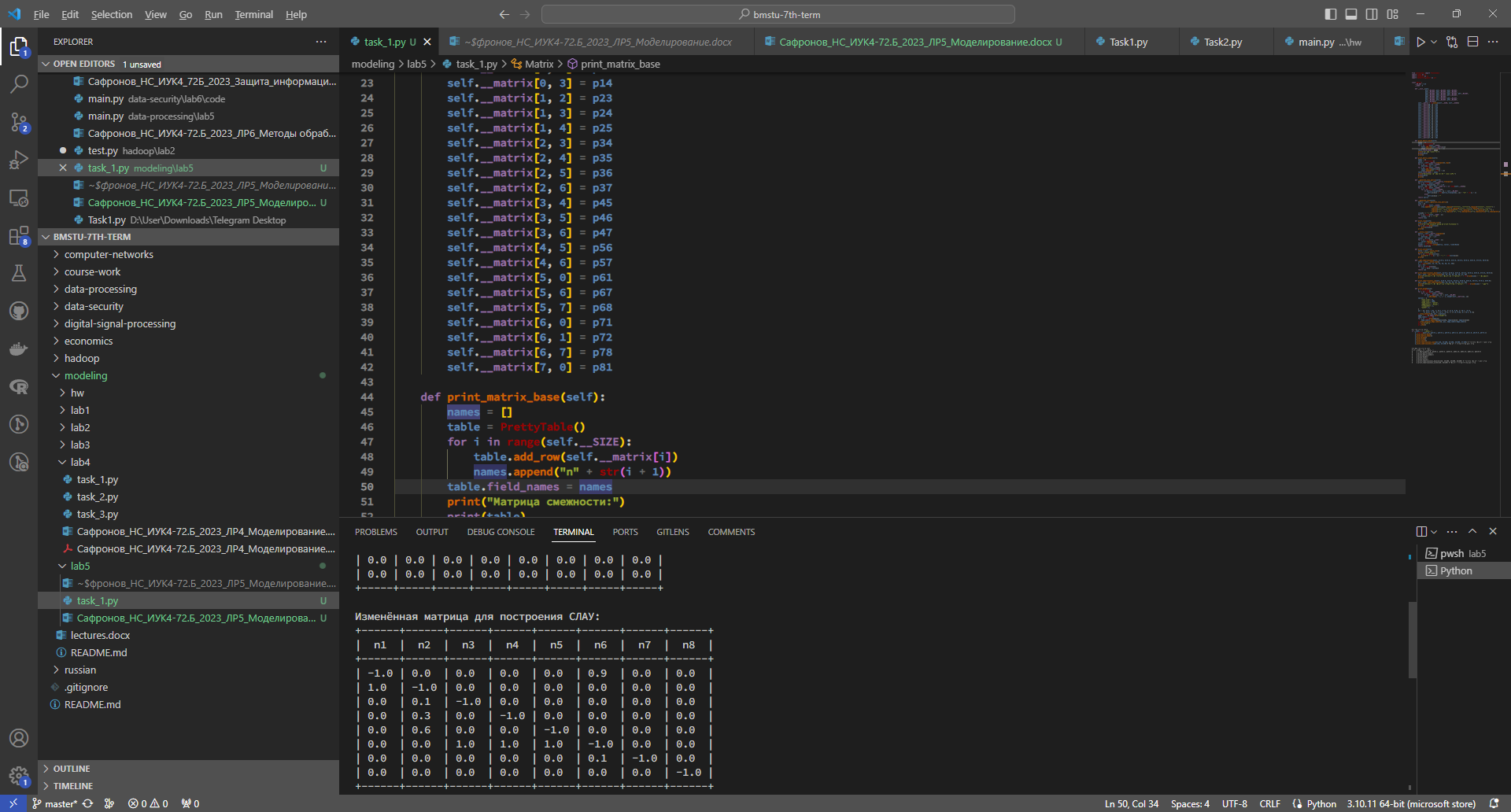
**Оценка трудоёмкости алгоритма по методу теории марковских цепей**



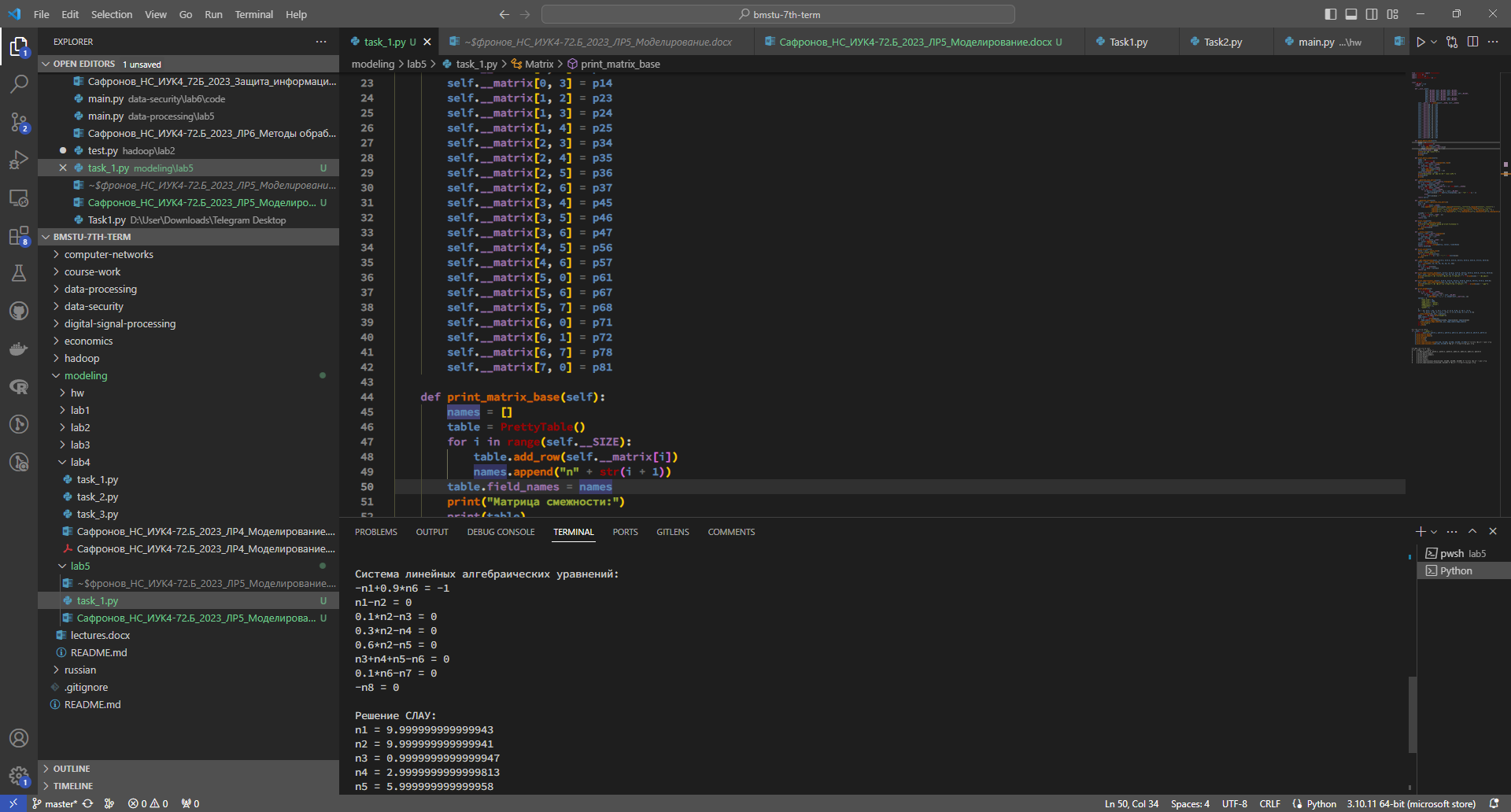
**Рисунок 1 -** Граф задания



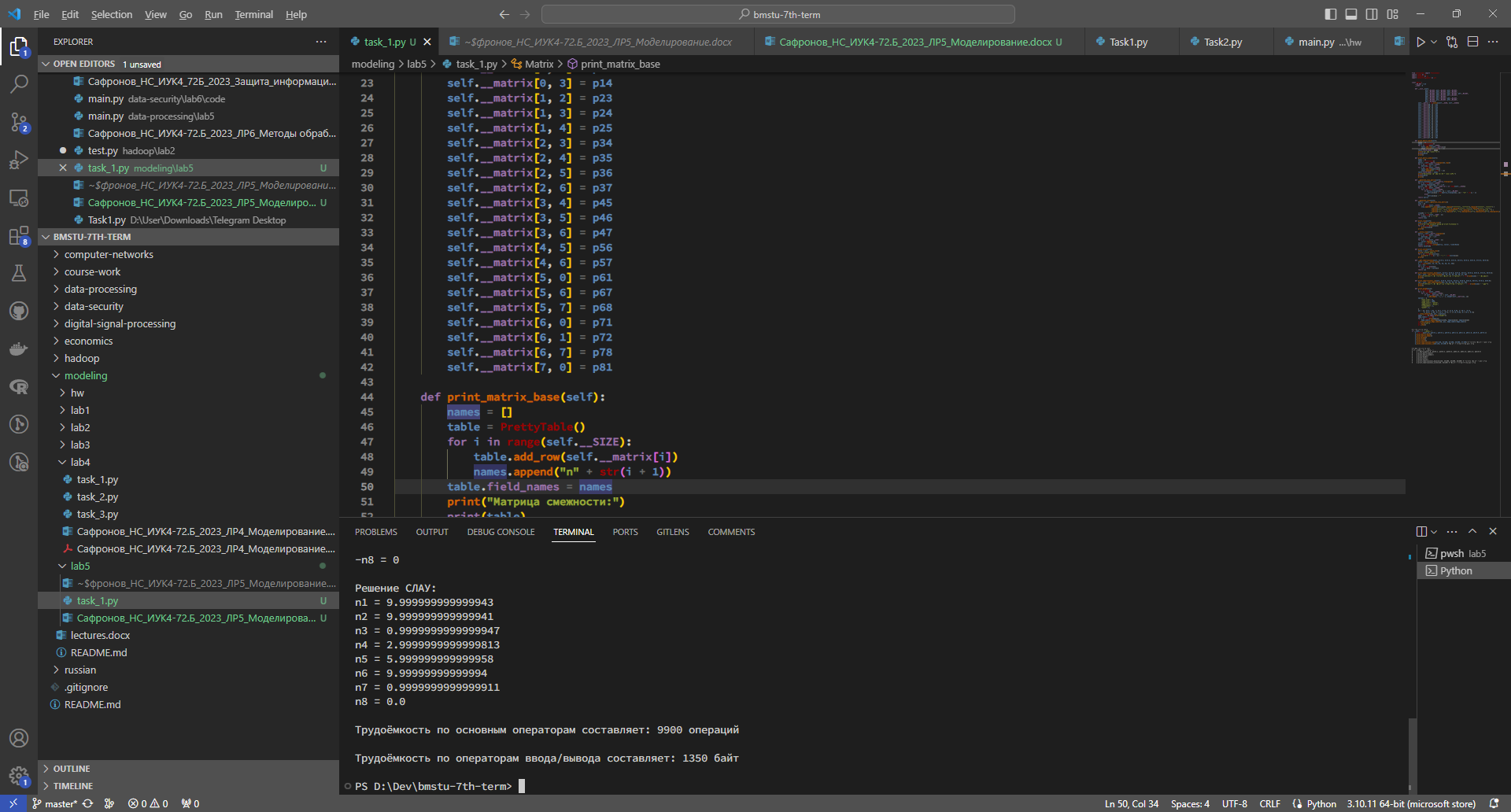
**Рисунок 2 -** Матрица смежности графа



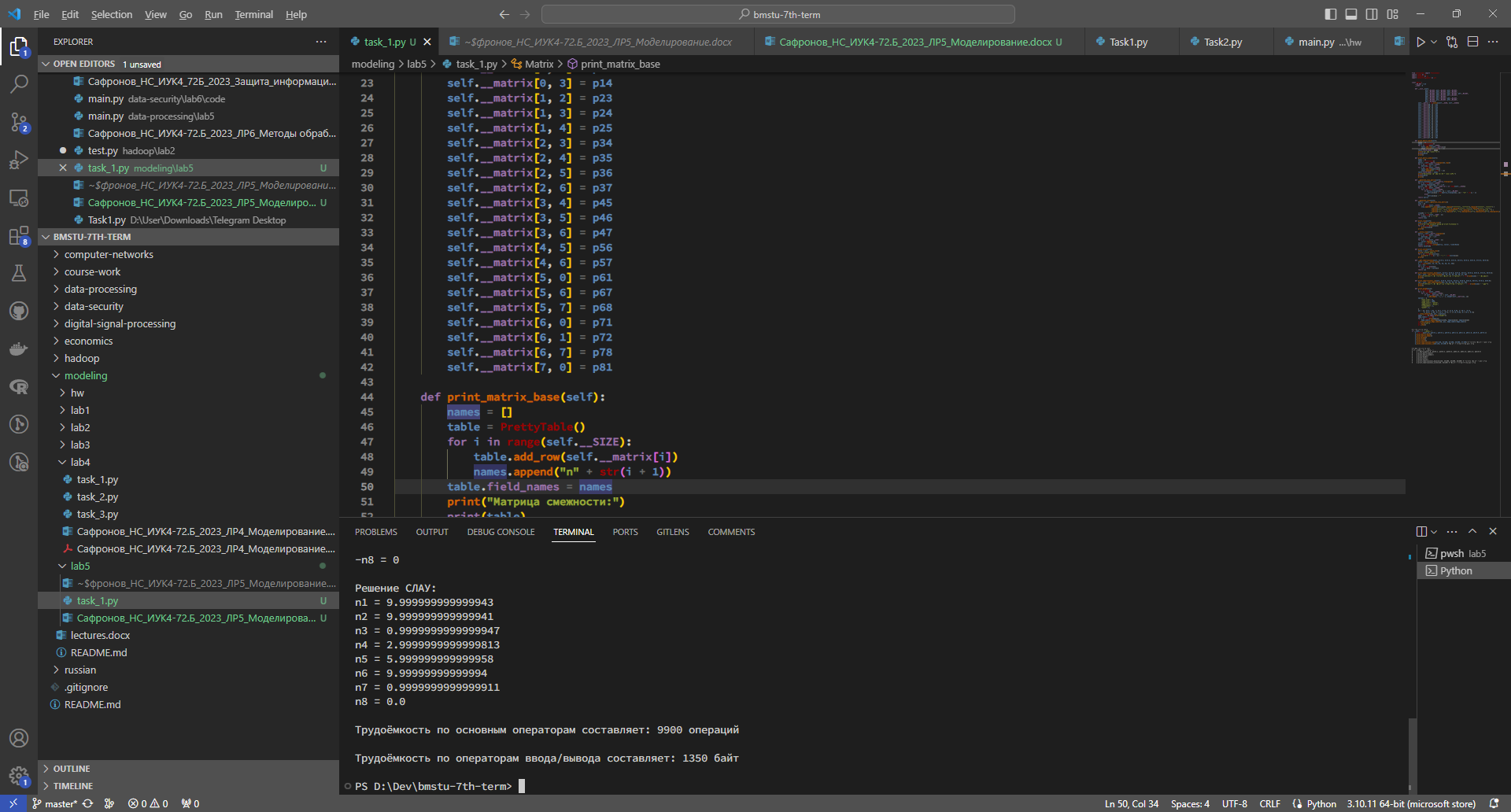
**Рисунок 3 -** Преобразованная матрица смежности для построения СЛАУ



**Рисунок 4 -** Полученная СЛАУ

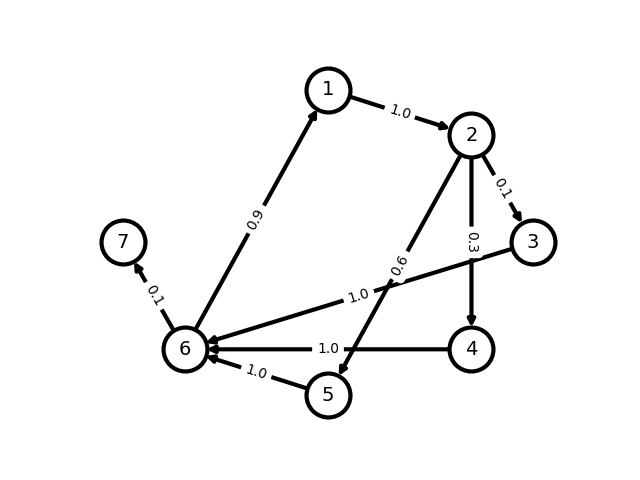


**Рисунок 5 -** Решение полученной СЛАУ

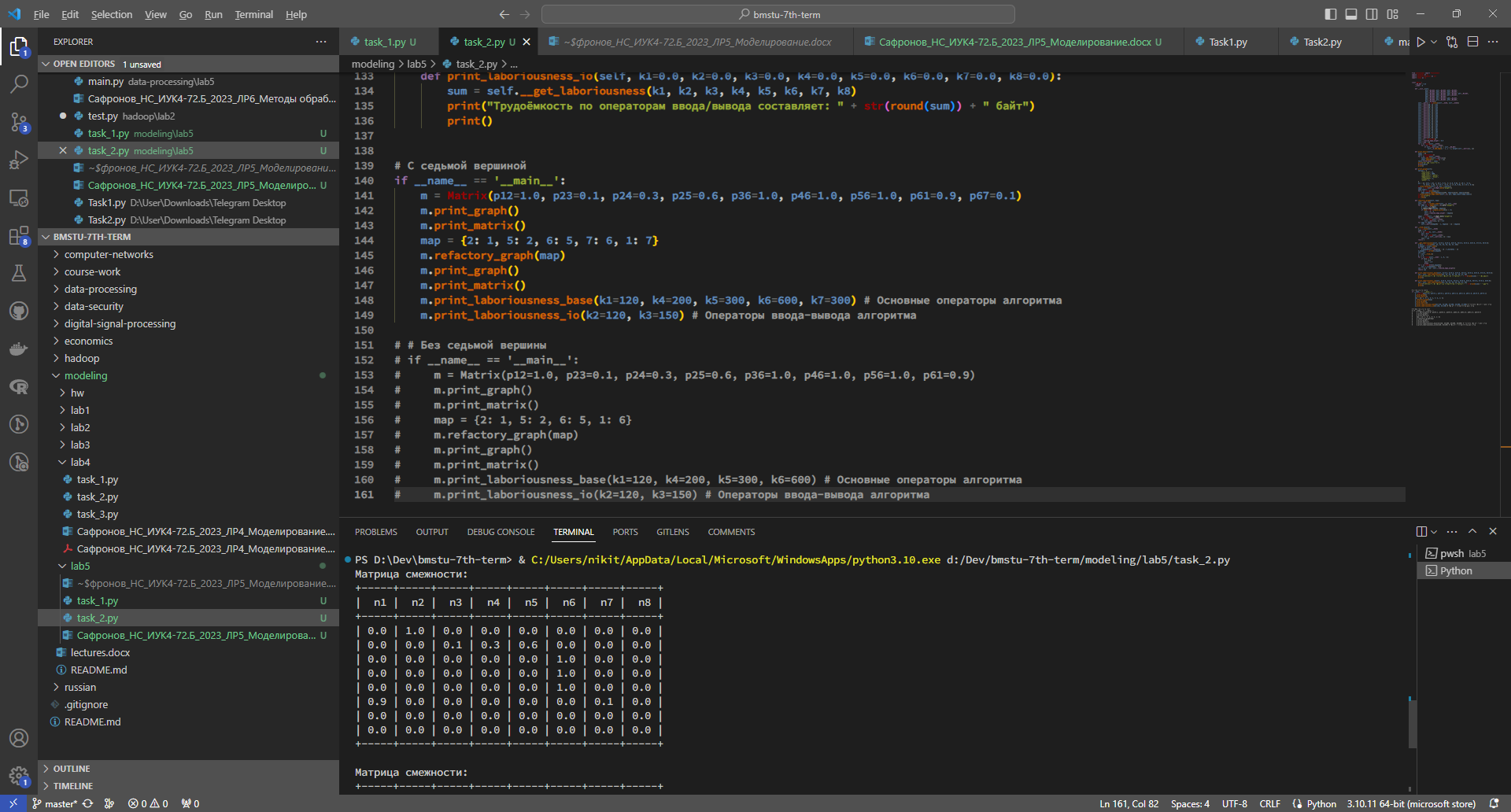


**Рисунок 6 -** Результат вычисления трудоёмкости

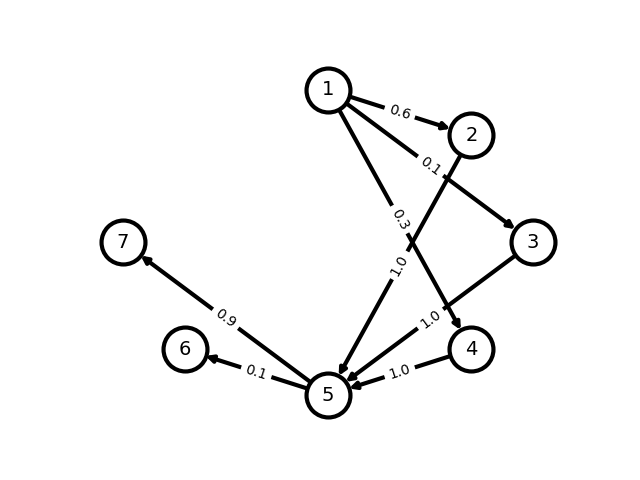
**Оценка трудоёмкости алгоритма сетевым методом**



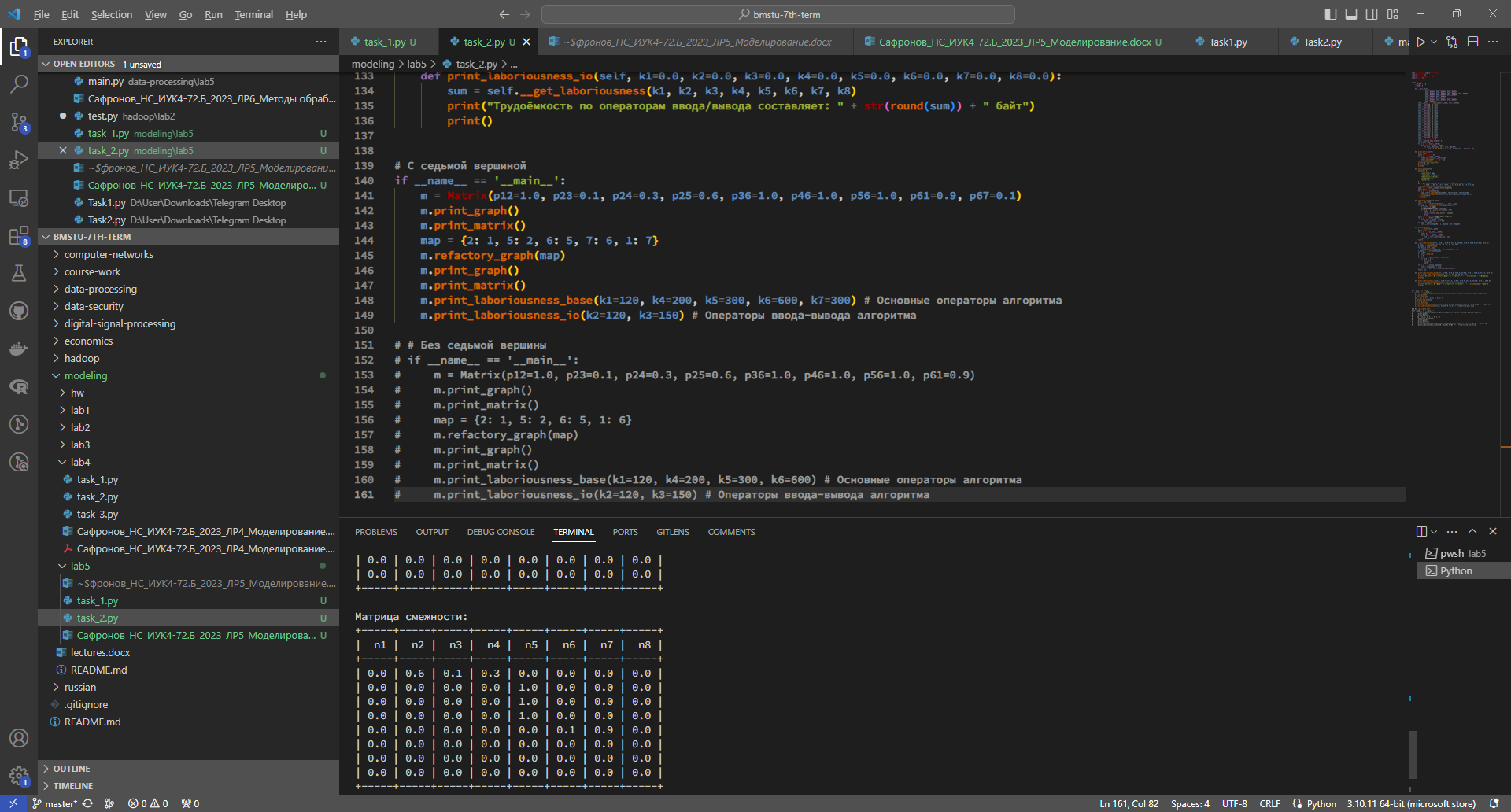
**Рисунок 7 -** Граф задания



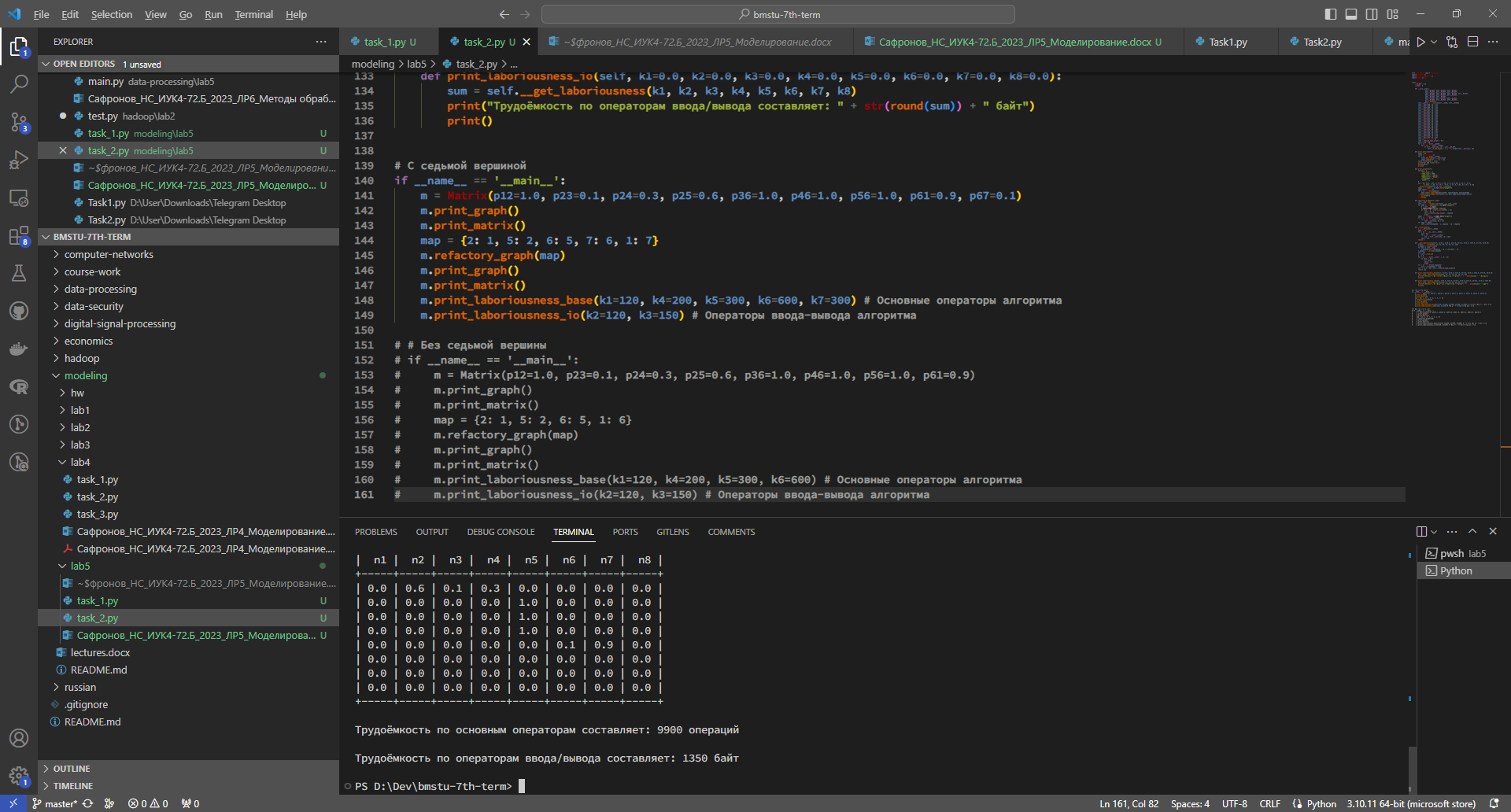
**Рисунок 8 -** Матрица смежности графа



**Рисунок 9 -** Преобразованный граф



**Рисунок 10 -** Матрица смежности преобразованного графа



**Рисунок 11 -** Результат вычисления трудоёмкости

Полученные результаты вычислений при помощи метода теории марковских цепей, а также сетевым методом тождественны. Из этого можно сделать вывод, что результаты исчислений трудоёмкостей верны.

**Вывод:** в ходе выполнения лабораторной работы были сформированы практические навыки технологии математического моделирования вычислительных алгоритмов, моделирования вычислительного алгоритма для оценки его трудоемкости.

**ПРИЛОЖЕНИЯ**

**Листинг программы**

**Задание 1**

from prettytable import PrettyTable

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

class Matrix:

\_\_NO\_WAY = 0.0

\_\_SIZE = 8

def \_\_init\_\_(self,

p12=\_\_NO\_WAY, p13=\_\_NO\_WAY, p14=\_\_NO\_WAY,

p23=\_\_NO\_WAY, p24=\_\_NO\_WAY, p25=\_\_NO\_WAY,

p34=\_\_NO\_WAY, p35=\_\_NO\_WAY, p36=\_\_NO\_WAY, p37=\_\_NO\_WAY,

p45=\_\_NO\_WAY, p46=\_\_NO\_WAY, p47=\_\_NO\_WAY,

p56=\_\_NO\_WAY, p57=\_\_NO\_WAY,

p61=\_\_NO\_WAY, p67=\_\_NO\_WAY, p68=\_\_NO\_WAY,

p71=\_\_NO\_WAY, p72=\_\_NO\_WAY, p78=\_\_NO\_WAY,

p81=\_\_NO\_WAY):

self.\_\_matrix = np.zeros((self.\_\_SIZE, self.\_\_SIZE))

self.\_\_matrix[0, 1] = p12

self.\_\_matrix[0, 2] = p13

self.\_\_matrix[0, 3] = p14

self.\_\_matrix[1, 2] = p23

self.\_\_matrix[1, 3] = p24

self.\_\_matrix[1, 4] = p25

self.\_\_matrix[2, 3] = p34

self.\_\_matrix[2, 4] = p35

self.\_\_matrix[2, 5] = p36

self.\_\_matrix[2, 6] = p37

self.\_\_matrix[3, 4] = p45

self.\_\_matrix[3, 5] = p46

self.\_\_matrix[3, 6] = p47

self.\_\_matrix[4, 5] = p56

self.\_\_matrix[4, 6] = p57

self.\_\_matrix[5, 0] = p61

self.\_\_matrix[5, 6] = p67

self.\_\_matrix[5, 7] = p68

self.\_\_matrix[6, 0] = p71

self.\_\_matrix[6, 1] = p72

self.\_\_matrix[6, 7] = p78

self.\_\_matrix[7, 0] = p81

def print\_matrix\_base(self):

names = []

table = PrettyTable()

for i in range(self.\_\_SIZE):

table.add\_row(self.\_\_matrix[i])

names.append("n" + str(i + 1))

table.field\_names = names

print("Матрица смежности:")

print(table)

print()

def print\_matrix\_modav(self):

names = []

table = PrettyTable()

matrix = self.\_\_matrix.transpose().copy()

for i in range(self.\_\_SIZE):

matrix[i, i] = -1.0

for i in range(self.\_\_SIZE):

table.add\_row(matrix[i])

names.append("n" + str(i + 1))

table.field\_names = names

print("Изменённая матрица для построения СЛАУ:")

print(table)

print()

def \_\_generate\_slau\_matrix(self):

matrix\_transpose = self.\_\_matrix.transpose()

for i in range(self.\_\_SIZE):

matrix\_transpose[i, i] = -1.0

matrix = [[str()] \* self.\_\_SIZE for i in range(self.\_\_SIZE)]

for i in range(self.\_\_SIZE):

for j in range(self.\_\_SIZE):

if matrix\_transpose[i, j] != self.\_\_NO\_WAY:

matrix[i][j] = str(matrix\_transpose[i, j]) + "\*n" + str(j + 1)

else:

matrix[i][j] = ""

return matrix

def \_\_generate\_slau(self):

matrix = self.\_\_generate\_slau\_matrix()

slau = []

for i in range(self.\_\_SIZE):

slau.append("+".join(matrix[i]).replace("++++++++", "+++++++").replace("+++++++", "++++++") \

.replace("++++++", "+++++").replace("+++++", "++++").replace("++++", "+++") \

.replace("+++", "++").replace("++", "+").replace("+-", "-").replace("1.0", "") \

.replace("-\*", "-").replace("+\*", "+").removeprefix("+").removesuffix("+").removeprefix("\*"))

slau[0] += " = -1"

for i in range(self.\_\_SIZE - 1):

slau[i + 1] += " = 0"

return slau

def print\_slau(self):

slau = self.\_\_generate\_slau()

print("Система линейных алгебраических уравнений:")

for i in range(len(slau)):

print(slau[i])

print()

def \_\_solve\_slau(self):

matrix = self.\_\_matrix.transpose()

for i in range(self.\_\_SIZE):

matrix[i, i] = -1.0

vector = [-1.0]

for i in range(self.\_\_SIZE - 1):

vector.append(0.0)

vector = np.array(vector)

answer = np.linalg.lstsq(matrix, vector, rcond=None)

return answer[0]

def print\_sovle(self):

sovle = self.\_\_solve\_slau()

print("Решение СЛАУ:")

for i in range(len(sovle)):

print("n" + str(i + 1) + " = " + str(sovle[i]))

print()

def \_\_get\_laboriousness(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0, k5=0.0, k6=0.0, k7=0.0, k8=0.0):

sovle = self.\_\_solve\_slau()

k = np.array((k1, k2, k3, k4, k5, k6, k7, k8))

sum = 0.0

for i in range(len(k)):

sum += k[i] \* sovle[i]

return sum

def print\_laboriousness\_base(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0, k5=0.0, k6=0.0, k7=0.0, k8=0.0):

sum = self.\_\_get\_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)

print("Трудоёмкость по основным операторам составляет: " + str(round(sum)) + " операций")

print()

def print\_laboriousness\_io(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0, k5=0.0, k6=0.0, k7=0.0, k8=0.0):

sum = self.\_\_get\_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)

print("Трудоёмкость по операторам ввода/вывода составляет: " + str(round(sum)) + " байт")

print()

def print\_graph(self):

G = nx.DiGraph()

for i in range(self.\_\_SIZE):

for j in range(self.\_\_SIZE):

if self.\_\_matrix[i, j] != self.\_\_NO\_WAY:

G.add\_edge(i + 1, j + 1, weight=self.\_\_matrix[i, j])

options = {

"font\_size": 14,

"node\_size": 1000,

"node\_color": "white",

"edgecolors": "black",

"linewidths": 3,

"width": 3,

}

pos = {1: (0.0, 1.0), 2: (0.7, 0.7), 3: (1.0, 0.0), 4: (0.7, -0.7),

5: (0.0, -1.0), 6: (-0.7, -0.7), 7: (-1.0, 0.0), 8: (-0.7, 0.7)}

nx.draw\_networkx(G, pos, \*\*options)

edges = list(G.edges.data("weight"))

edge\_labels = dict()

for i in range(len(edges)):

edge\_labels.update({(edges[i][0], edges[i][1]): edges[i][2]})

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels)

plt.axis("off")

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

m = Matrix(p12=1.0, p23=0.1, p24=0.3, p25=0.6, p36=1.0, p46=1.0, p56=1.0, p61=0.9, p67=0.1)

m.print\_matrix\_base()

m.print\_matrix\_modav()

m.print\_graph()

m.print\_slau()

m.print\_sovle()

m.print\_laboriousness\_base(k1=120, k4=200, k5=300, k6=600, k7=300)

m.print\_laboriousness\_io(k2=120, k3=150)

**Задание 2**

from prettytable import PrettyTable

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

class Matrix:

\_\_NO\_WAY = 0.0

\_\_SIZE = 8

def \_\_init\_\_(self,

p12=\_\_NO\_WAY, p13=\_\_NO\_WAY, p14=\_\_NO\_WAY,

p23=\_\_NO\_WAY, p24=\_\_NO\_WAY, p25=\_\_NO\_WAY,

p34=\_\_NO\_WAY, p35=\_\_NO\_WAY, p36=\_\_NO\_WAY, p37=\_\_NO\_WAY,

p45=\_\_NO\_WAY, p46=\_\_NO\_WAY, p47=\_\_NO\_WAY,

p56=\_\_NO\_WAY, p57=\_\_NO\_WAY,

p61=\_\_NO\_WAY, p67=\_\_NO\_WAY, p68=\_\_NO\_WAY,

p71=\_\_NO\_WAY, p72=\_\_NO\_WAY, p78=\_\_NO\_WAY,

p81=\_\_NO\_WAY):

self.\_\_matrix = np.zeros((self.\_\_SIZE, self.\_\_SIZE))

self.\_\_matrix[0, 1] = p12

self.\_\_matrix[0, 2] = p13

self.\_\_matrix[0, 3] = p14

self.\_\_matrix[1, 2] = p23

self.\_\_matrix[1, 3] = p24

self.\_\_matrix[1, 4] = p25

self.\_\_matrix[2, 3] = p34

self.\_\_matrix[2, 4] = p35

self.\_\_matrix[2, 5] = p36

self.\_\_matrix[2, 6] = p37

self.\_\_matrix[3, 4] = p45

self.\_\_matrix[3, 5] = p46

self.\_\_matrix[3, 6] = p47

self.\_\_matrix[4, 5] = p56

self.\_\_matrix[4, 6] = p57

self.\_\_matrix[5, 0] = p61

self.\_\_matrix[5, 6] = p67

self.\_\_matrix[5, 7] = p68

self.\_\_matrix[6, 0] = p71

self.\_\_matrix[6, 1] = p72

self.\_\_matrix[6, 7] = p78

self.\_\_matrix[7, 0] = p81

self.\_\_G = nx.DiGraph()

self.\_\_removed\_edge\_weight = 0.0

self.\_\_map = {}

for i in range(self.\_\_SIZE):

for j in range(self.\_\_SIZE):

if self.\_\_matrix[i, j] != self.\_\_NO\_WAY:

self.\_\_G.add\_edge(i + 1, j + 1, weight=self.\_\_matrix[i, j])

def print\_matrix(self):

names = []

table = PrettyTable()

for i in range(self.\_\_SIZE):

table.add\_row(self.\_\_matrix[i])

names.append("n" + str(i + 1))

table.field\_names = names

print("Матрица смежности:")

print(table)

print()

def print\_graph(self):

options = {

"font\_size": 14,

"node\_size": 1000,

"node\_color": "white",

"edgecolors": "black",

"linewidths": 3,

"width": 3,

}

pos = {1: (0.0, 1.0), 2: (0.7, 0.7), 3: (1.0, 0.0), 4: (0.7, -0.7),

5: (0.0, -1.0), 6: (-0.7, -0.7), 7: (-1.0, 0.0), 8: (-0.7, 0.7)}

nx.draw\_networkx(self.\_\_G, pos, \*\*options)

edges = list(self.\_\_G.edges.data("weight"))

edge\_labels = dict()

for i in range(len(edges)):

edge\_labels.update({(edges[i][0], edges[i][1]): edges[i][2]})

nx.draw\_networkx\_edge\_labels(self.\_\_G, pos, edge\_labels=edge\_labels)

plt.axis("off")

plt.show()

def refactory\_graph(self, map):

self.\_\_map = map

self.\_\_G = nx.relabel\_nodes(self.\_\_G, self.\_\_map)

for edge in nx.edges(self.\_\_G).data("weight"):

G = self.\_\_G.copy()

G.remove\_edge(edge[0], edge[1])

if len(list(nx.simple\_cycles(G))) == 0:

self.\_\_G = G

self.\_\_removed\_edge\_weight = edge[2]

break

edges = list(self.\_\_G.edges.data("weight"))

for i in range(self.\_\_SIZE):

for j in range(self.\_\_SIZE):

self.\_\_matrix[i, j] = 0.0

for edge in edges:

self.\_\_matrix[edge[0] - 1, edge[1] - 1] = edge[2]

def \_\_find\_n(self):

n = np.zeros(self.\_\_SIZE)

n[0] = 1.0

for i in range(1, self.\_\_SIZE):

sum = 0.0

for j in range(self.\_\_SIZE):

sum += self.\_\_matrix[j, i] \* n[j]

n[i] = sum

return n

def \_\_get\_laboriousness(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0, k5=0.0, k6=0.0, k7=0.0, k8=0.0):

k\_base = np.array((k1, k2, k3, k4, k5, k6, k7, k8))

k\_mapped = k\_base.copy()

for key in self.\_\_map:

k\_mapped[self.\_\_map[key] - 1] = k\_base[key - 1]

k\_mapped = np.array(k\_mapped)

sum = 0.0

n = self.\_\_find\_n()

p = 1.0

for i in range(self.\_\_SIZE - 1, 0, -1):

if n[i] != 0.0:

p = n[i]

n[i] = 1.0

break

for i in range(len(k\_mapped)):

sum += k\_mapped[i] \* n[i]

sum /= (1.0 - (p \* self.\_\_removed\_edge\_weight))

return sum

def print\_laboriousness\_base(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0, k5=0.0, k6=0.0, k7=0.0, k8=0.0):

sum = self.\_\_get\_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)

print("Трудоёмкость по основным операторам составляет: " + str(round(sum)) + " операций")

print()

def print\_laboriousness\_io(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0, k5=0.0, k6=0.0, k7=0.0, k8=0.0):

sum = self.\_\_get\_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)

print("Трудоёмкость по операторам ввода/вывода составляет: " + str(round(sum)) + " байт")

print()

if \_\_name\_\_ == '\_\_main\_\_':

m = Matrix(p12=1.0, p23=0.1, p24=0.3, p25=0.6, p36=1.0, p46=1.0, p56=1.0, p61=0.9, p67=0.1)

m.print\_graph()

m.print\_matrix()

map = {2: 1, 5: 2, 6: 5, 7: 6, 1: 7}

m.refactory\_graph(map)

m.print\_graph()

m.print\_matrix()

m.print\_laboriousness\_base(k1=120, k4=200, k5=300, k6=600, k7=300)

m.print\_laboriousness\_io(k2=120, k3=150)