**Санкт-Петербургский государственный университет**

**Факультет прикладной математики и процессов управления**

**Лабораторная работа №5**

**Решение задачи о коммивояжере с помощью алгоритма имитации отжига**

|  |  |  |
| --- | --- | --- |
| Студентка гр. Б15-ПУ |  | Гладкая М.В. |
| Преподаватель |  | Дик А.Г. |

Санкт-Петербург

2023 г.

**Оглавление**

1. **Цель работы ………………………………………………………………3**
2. **Задача………………………………………………………………………3**
3. **Теоритическая часть……………………………………………………..3**
4. **Описание программы ……………………………………………………3**
   1. **Список используемых библиотек……...………………………...…4**
   2. **Описание функций и классов..…………...…………………………4**
   3. **Общий ход программы………………………………………………5**
5. **Рекомендации программиста……………………………………………5**
6. **Контрольный пример…………………………………………………….6**
7. **Результаты работы алгоритма на заданном графе……..…………....8**
8. **Вывод…………………………………………..………………………..….9**
9. **Приложения………………………………………………………………10**
10. **Цель работы**

Исследование алгоритма оптимизации метода имитации для решения задачи о коммивояжере на примере графов.

1. **Задачи**

* Изучить алгоритм имитации отжига
* Написать программу поиска кратчайшего гамильтонова цикла в графе с помощью алгоритма имитации отжига
* Протестировать программу
* Реализовать GUI
* Найти кратчайший гамильтонов цикл для заданного графа

1. **Теоритическая часть**

Имитация отжига – эвристический алгоритм, имитирующий физический процесс кристаллизации вещества при понижающейся температуре.

Метод имитации отжига в исходной формулировке заключается в нахождении замкнутой кривой минимальной длины, соединяющей заданный набор точек на плоскости.

В нашем случае набором точек на плоскости будет являться взвешенный ориентированный граф.

Шаги алгоритма:

1. Случайно выбрать гамильтонов цикл на заданном графе,
2. Случайно выбрать два числа – индексы пути обхода вершин выбранного цикла, а затем поменять эти вершины местами в пути обхода,
3. Установить начальное значение температуры ***t*** *= 100, a = (0,1),*
4. Вычислить длины исходного (***L1***) и получившегося путей (***L2***),
5. Вычислить ***L*** *= L2 – L1,*
6. Если *L < 0*, то выбираем получившийся путь как исходный для следующей итерации цикла,

Если *L >= 0*, то генерируем случайное число **p** от 0 до 100, рассчитываем **p\*** = 100 \* (e^(-L/t)),

* 1. Если *p > p\*,* то оставляем прежний путь как исходный для следующей итерации цикла,

Если *p <= p\*,* то выбираем получившийся путь как исходный для следующей итерации цикла,

1. Снижаем температуру *t = at,*
2. Если выполняется критерий останова, то завершаем работу алгоритма, иначе переходим к шагу 2.
3. **Описание программы**

Программный код написан на языке Python.С кодом можно ознакомиться по ссылке <https://github.com/9Neechan/simulatedAnnealingMethod.git> .

* 1. **Список используемых библиотек**

Список используемых библиотек представлен в Таблице 1.

*Таблица 1. Список используемых библиотек*

|  |  |
| --- | --- |
| **Название** | **Описание** |
| PySimpleGUI | Реализация GUI |
| networkx | Работа с графами, как со структурой |
| matplotlib | Отрисовка графиков |
| random | Генерация рандомных значений |
| math | Использование математических функций |

* 1. **Описание функций и классов**

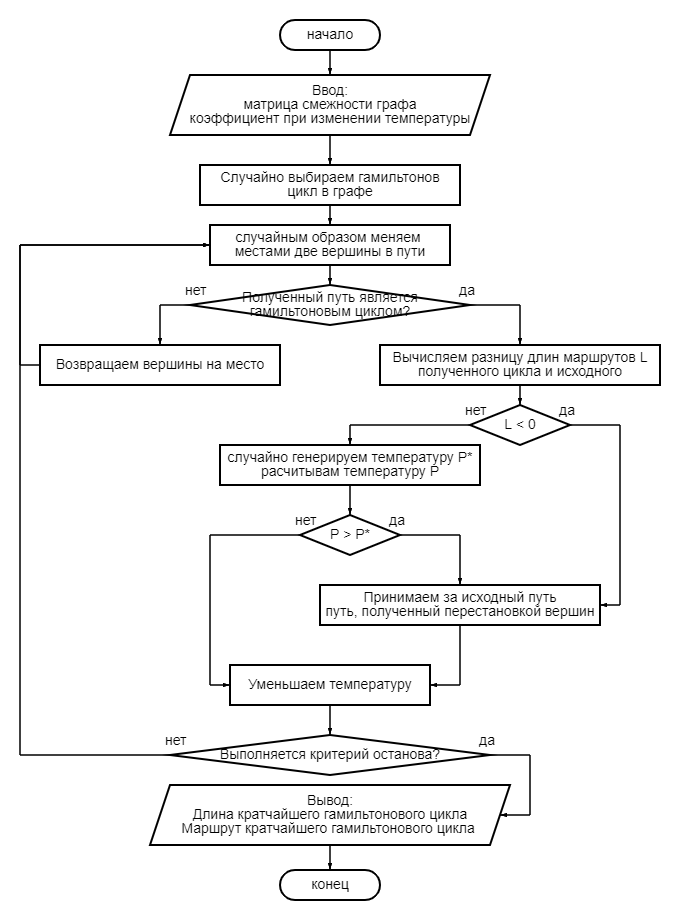
В таблице 2 представлено описание функций файла algorithm.py.

*Таблица 2. Функции algorithm.py*

|  |  |  |
| --- | --- | --- |
| **Имя функции** | **Входные данные** | **Описание функции** |
| create\_multigraph\_struct() | graph – матрица смежности графа | Создает экземпляр класса мультиграфа из библиотеки networkx на основе переданной матрицы смежности графа |
| create\_graph\_struct() | path\_arr - путь  graph – матрица смежности графа | Создает экземпляр класса орграфа из библиотеки networkx на основе переданного пути |
| make\_plt() | flag – флаг, казывающий на то, какой вид графа создавать,  path\_arr - путь,  graph – матрица смежности графа,  i – индекс пути в списке путей | Отрисовывает граф по переданным параметрам с помощью matplotlib и сохраняет в папку pictires |
| draw\_graph() | im\_arr – массив, в который будут сохраняться изображения графов  paths\_arr – массив путей обхода  graph – матрица смежности графа | Отрисовывает изображения результатов работы алгоритма, которые будут использоваться в GUI |
| generate\_random\_path() | graph – матрица смежности графа | Генерирует рандомно гамильтонов цикл в графе и считает его длину |
| check\_edges() | path\_arr - путь,  graph – матрица смежности графа, | Проверяет является ли путь гамильтоновым циклом в графе |
| simulated\_annealing\_method () | graph – матрица смежности графа  alpha – коэффициент при изменении температуры | Реализует работу метода имитации отжига на ориентированном взвешенном графе |

* 1. **Общий ход программы**

На рисунке 1 изображен ход подпрограммы, реализующей работу метода имитации отжига.



*Рисунок 1. Блок-схема подпрограммы, реализующей работу метода имитации отжига*

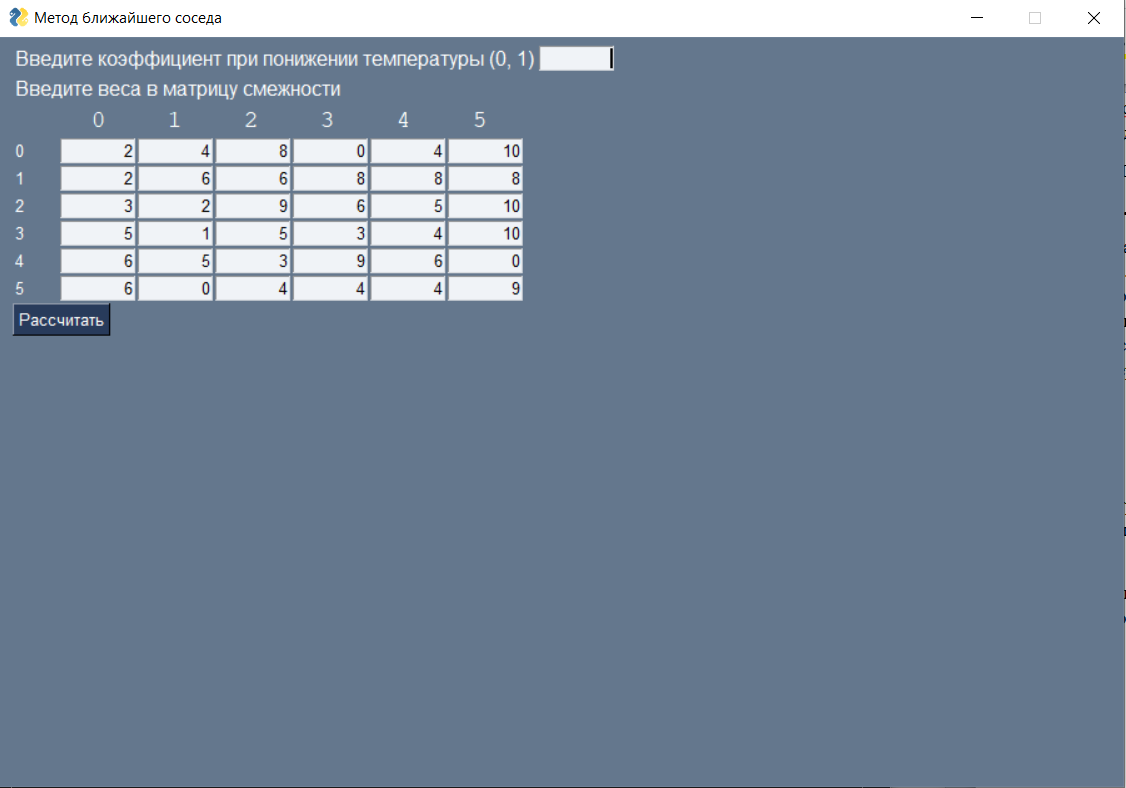
1. **Рекомендации программиста**

Для запуска программы необходима 64-битная операционная система Windows и python не ниже 3.9. Для работы с кодом необходима IDE PyCharm или другая любая среда разработки для python.

Импортируйте библиотеки из пункта 4.1.

1. **Контрольный пример**

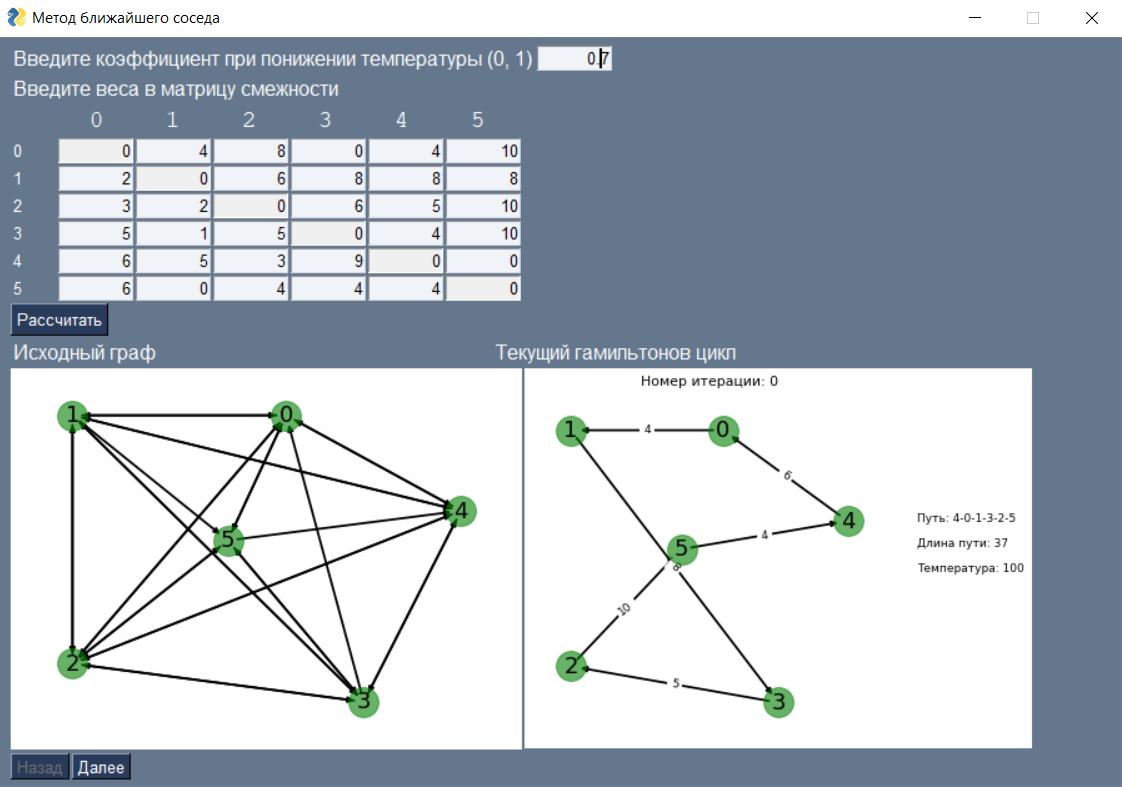
Запустите программу. Перед вами откроется начальное окно программы (Рис. 2). Введите коэффициент при изменении температуры и веса ребер в матрицу смежности графа, для которого хотите произвести расчеты или оставьте автоматически сгенерированные значения. Затем нажмите кнопку «Рассчитать» для расчета наикратчайших гамильтоновых циклов, содержащихся в графе с помощью метода ближайшего соседа.



*Рисунок 2. Начальное окно*

По окончании расчетов будут выведены изображения графа, построенного на основе матрицы смежности и графы гамильтоновых циклов на каждой итерации работы алгоритма.

Чтобы пролистывать между изображениями графов нажимайте на кнопки «Далее» и «Назад». На рисунке 3 изображен один из результатов работы программы.

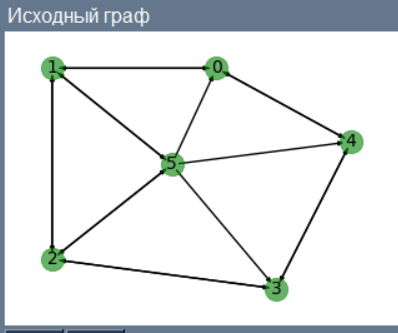


*Рисунок 3. Окно программы после нажатия кнопки «Рассчитать»*

Чтобы сделать расчеты для другого графа, введите новые значения в таблицу смежности и нажмите кнопку «Рассчитать».

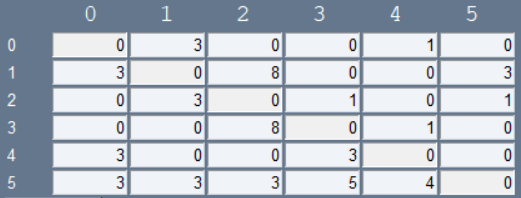
1. **Результаты работы алгоритма на заданном графе**

Расчеты были сделаны для графа следующего вида (рисунок 4).



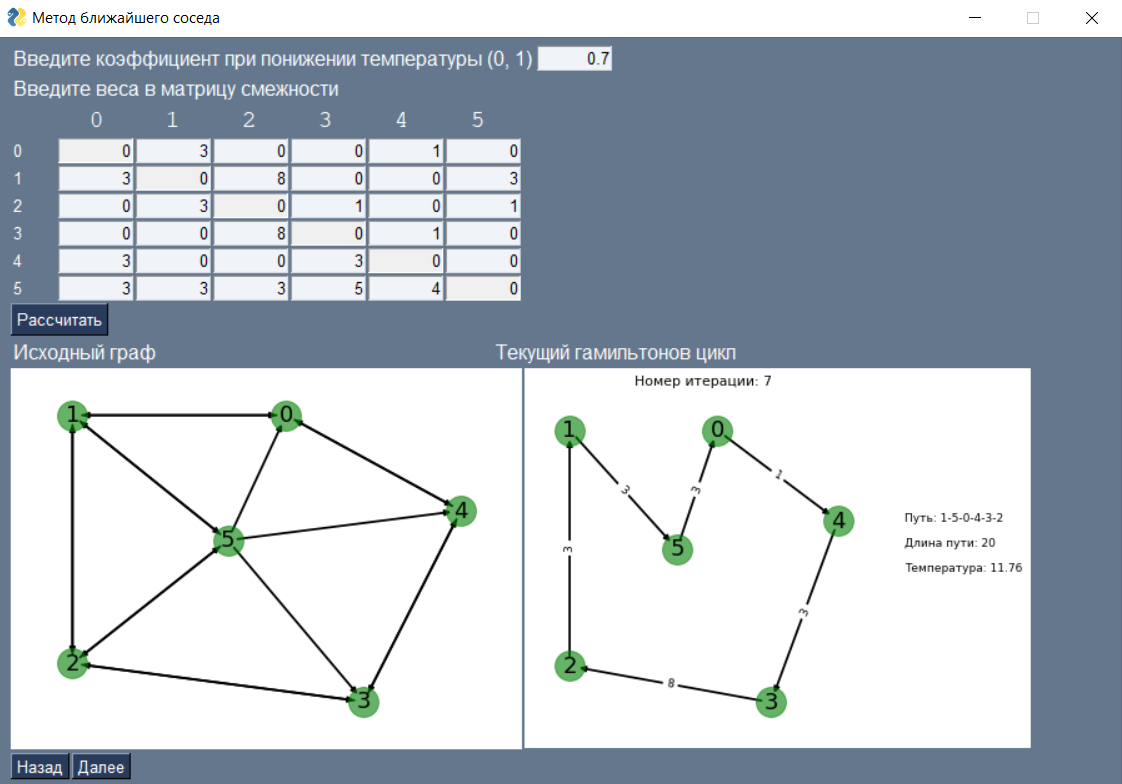
*Рисунок 4. Заданный орграф*

Матрица смежности имеет следующий вид (рисунок 5).



*Рисунок 5. Матрица смежности*

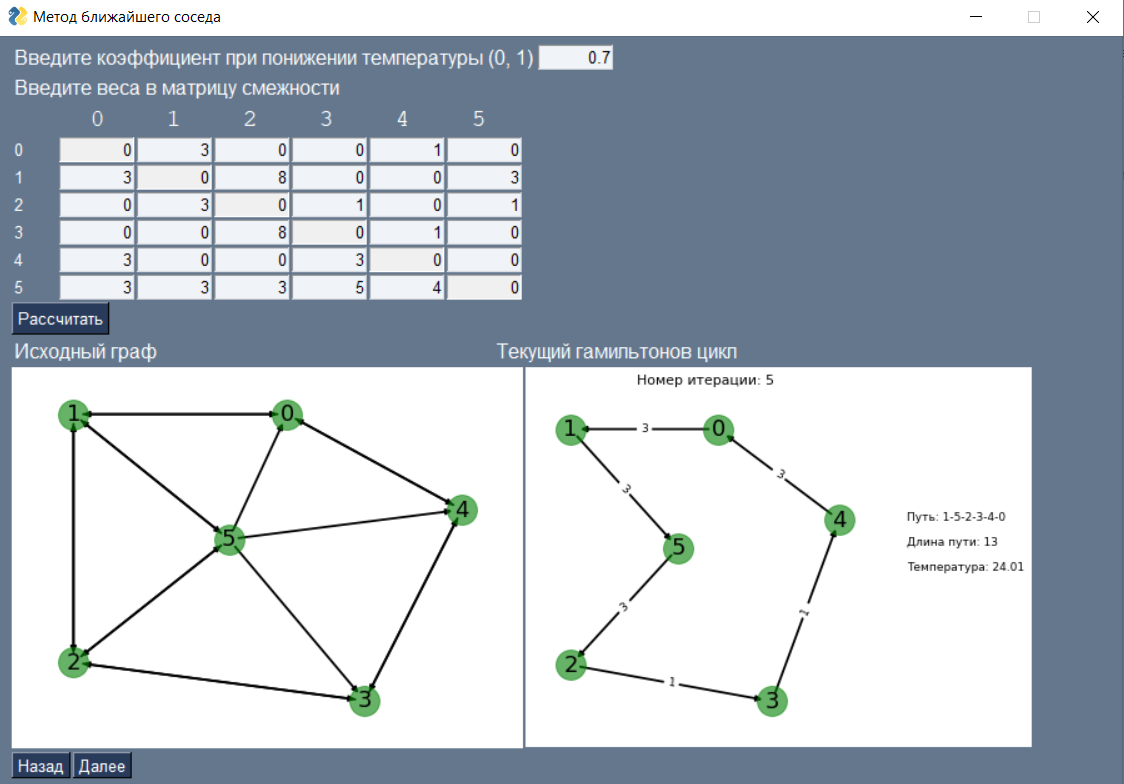
По результатам работы алгоритма длина кратчайшего гамильтонового цикла равна 20 (рисунок 6).



*Рисунок 6. Результат работы программы*

Но если пролистать итерации цикла, то можно заметить, что были найдены так же и более короткие пути с длиной 13 (рисунок 7). Это свидетельствует о том, что точность алгоритма приблизительная и сильно зависит от генерации случайных значений.

Алгоритм имитации отжига имеет вероятностную природу, поэтому может застрять в локальном минимуме. Поэтому стоит сделать несколько запусков и обращать внимание, к какому минимальному значению сходятся результаты.



*Рисунок 6. Одна из итераций работы программы*

От коэффициента при понижении температуры зависит, на сколько быстро она будет снижаться. Чем больше коэффициент, тем медленнее падает температура, тем больше итераций будет высокий шанс на выбор более длинного маршрута, тем больше вероятность найти минимальный маршрут.

Сравнение алгоритма имитации отжига с алгоритмом поиска ближайшего соседа:

Алгоритм поиска ближайшего соседа относится к жадным алгоритмам, что означает большее время работы программы. Так как алгоритм перебирает все вершины, то он зачастую находит одинаковые циклы, которые были получены при начале расчета от разных вершин. Падает качество ответа при увеличении количества вершин.

Алгоритм имитации отжига находит более оптимальные пути, чем алгоритм поиска ближайшего соседа. Это видно по результатам их работы: алгоритм отжига находил минимальный цикл со стоимостью 13, а алгоритм поиска ближайшего соседа - со стоимостью 14.

Таким образом, алгоритм имитации отжига работает быстрее и точнее.

1. **Вывод**

В ходе данной работы был изучен метод имитации отжига, написана программа поиска кратчайшего гамильтонова цикла в орграфе с помощью метода имитации отжига, реализован GUI, найдены кратчайшие гамильтоновы циклы для заданного графа.

**Приложения**

Приложение А. algorithm.py

1. **import** matplotlib.pyplot as plt
2. **import** networkx as nx
3. **import** random
4. **import** math

7. **def** create\_multigraph\_struct(graph):
8. G **=** nx.MultiDiGraph()
9. **for** i **in** range(len(graph)):
10. **for** j **in** range(len(graph[0])):
11. **if** graph[i][j] !**=** 0 **and** i !**=** j:
12. G.add\_edge(f'{i}', f'{j}', weight**=**graph[i][j])
13. **return** G

16. **def** create\_graph\_struct(path\_arr, graph):
17. G **=** nx.DiGraph()
18. data **=** []
19. **for** j **in** range(len(path\_arr)):
20. **if** j **+** 1 < len(path\_arr):
21. data.append(path\_arr[j **+** 1])
22. data.append(path\_arr[0])
23. **for** i **in** range(len(path\_arr)):
24. G.add\_edge(f'{path\_arr[i]}', f'{data[i]}', weight**=**graph[path\_arr[i]][data[i]])
25. **return** G

28. **def** make\_plt(flag, path\_arr, graph, i, path\_len, t):
29. """Рисует графы с помощью бибилиотеки matplotlib и сохраняет в папку pictures"""
30. pos **=** {'0': [0, 0.25],
31. '1': [**-**0.55, 0.25],
32. '2': [**-**0.55, **-**0.4],
33. '3': [0.2, **-**0.5],
34. '4': [0.45, 0],
35. '5': [**-**0.15, **-**0.0775903]}
36. path **=** ''
38. **if** flag **==** 'multi':
39. G **=** create\_multigraph\_struct(graph)
40. snapshot\_name **=** "pictures/initial.png"
41. **else**:
42. G **=** create\_graph\_struct(path\_arr, graph)
43. snapshot\_name **=** f"pictures/{i}.png"
44. path **=** f'{str(path\_arr[0])}-{str(path\_arr[1])}-' \
45. f'{str(path\_arr[2])}-{str(path\_arr[3])}-' \
46. f'{str(path\_arr[4])}-{str(path\_arr[5])}'
48. # nodes
49. nx.draw\_networkx\_nodes(G, pos, node\_size**=**700, node\_color**=**'green', alpha**=**0.6)
50. # edges
51. nx.draw\_networkx\_edges(G, pos, width**=**2)
52. # node labels
53. nx.draw\_networkx\_labels(G, pos, font\_size**=**20, font\_family**=**"sans-serif")
54. **if** flag **==** "basic":
55. # edge weight labels
56. edge\_labels **=** nx.get\_edge\_attributes(G, "weight")
57. nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels)
58. plt.title(f'Номер итерации: {i}')
59. plt.text(0.7, 0, f'Путь: {path}')
60. plt.text(0.7, **-**0.07, f'Длина пути: {path\_len}')
61. plt.text(0.7, **-**0.14, f'Температура: {t}')
62. ax **=** plt.gca()
63. ax.margins(0.08)
64. plt.axis("off")
65. plt.tight\_layout()
66. plt.savefig(snapshot\_name, dpi**=**65, bbox\_inches**=**'tight')
67. plt.close()

70. **def** draw\_graph(data, graph):
71. """Визуализация графов"""
72. flag **=** 'multi'
73. make\_plt(flag, data[0][1], graph, 0, data[0][0], data[0][3])
75. # рисуем стадии работы цикла
76. **for** i **in** range(len(data)):
77. flag **=** 'basic'
78. make\_plt(flag, data[i][1], graph, data[i][0], data[i][2], data[i][3])

81. **def** generate\_random\_path(graph):
82. """Генерирует рандомно гамильтонов цикл в графе и считает его длину"""
83. g\_flag **=** 0
84. path **=** []
85. leng **=** 0
86. **while** g\_flag **==** 0:
87. list **=** [0, 1, 2, 3, 4, 5]
88. path **=** []
89. leng **=** 0
90. v **=** random.choice(list)
91. path.append(v)
92. list.remove(v)
93. **for** i **in** range(1, len(graph)):
94. flag **=** 0
95. **while** flag **==** 0:
96. v **=** random.choice(list)
97. **if** graph[path[i**-**1]][v] !**=** 0:
98. path.append(v)
99. list.remove(v)
100. leng **+=** graph[path[i**-**1]][v]
101. flag **=** 1
102. **if** graph[path[len(graph)**-**1]][path[0]] !**=** 0:
103. leng **+=** graph[path[len(graph)**-**1]][path[0]]
104. g\_flag **=** 1
106. **return** path, leng

109. **def** check\_edges(path, graph):
110. """Проверяет является ли путь гамильтоновым циклом"""
111. leng **=** **-**1
112. **for** i **in** range(len(path)):
113. **if** i **==** len(path)**-**1:
114. **if** graph[path[i]][path[0]] **==** 0:
115. **return** False, leng
116. **else**:
117. leng **+=** graph[path[i]][path[0]]
118. **else**:
119. **if** graph[path[i]][path[i**+**1]] **==** 0:
120. **return** False, leng
121. **else**:
122. leng **+=** graph[path[i]][path[i**+**1]]
123. **return** True, leng

126. **def** simulated\_annealing\_method(graph, alpha):
127. data\_for\_pics **=** []
129. # находим рандомный цикл в графе
130. path, len\_path **=** generate\_random\_path(graph)
131. t **=** 100
132. i **=** 0
134. data\_for\_pics.append([i, path, len\_path, t])
136. **while** data\_for\_pics[i][1] !**=** data\_for\_pics[i**-**1][1] !**=** data\_for\_pics[i**-**2][2] !**=** data\_for\_pics[i**-**3][3] **or** i < 4:
137. path1 **=** path.copy()
138. len1\_path **=** 0
140. g\_flag **=** 0
141. **while** g\_flag **==** 0:
142. # генерируем индексы по которым будем менять вершины в пути
143. flag **=** 0
144. v1 **=** random.randint(0, 5)
145. v2 **=** **-**1
146. **while** flag **==** 0:
147. v2 **=** random.randint(0, 5)
148. **if** v2 !**=** v1:
149. flag **=** 1
150. path1[v1], path1[v2] **=** path1[v2], path1[v1]
152. # проверяем является ли новый путь гамильтоновым циклом
153. booll, new\_len **=** check\_edges(path1, graph)
154. **if** booll:
155. g\_flag **=** 1
156. len1\_path **=** new\_len
158. # вычисляем дельту длин маршрутов
159. delta\_l **=** len1\_path **-** len\_path
161. **if** delta\_l < 0:
162. path **=** path1
163. len\_path **=** len1\_path
164. **else**:
165. p\_rand **=** random.randint(0, 100)
166. p **=** 100 **\*** math.exp(**-**delta\_l**/**t)
167. **if** p > p\_rand:
168. path **=** path1
169. len\_path **=** len1\_path
171. i **+=** 1
172. data\_for\_pics.append([i, path, len\_path, round(t, 2)])
173. t **\*=** alpha
175. **for** el **in** data\_for\_pics:
176. print(el)
178. draw\_graph(data\_for\_pics, graph)
179. **return** data\_for\_pics

Приложение Б. GUI.py

1. **import** PySimpleGUI as psg
2. **from** random **import** randint
3. **import** algorithm
5. MAX\_ROWS, MAX\_COLS, COL\_HEADINGS **=** 6, 6, ('    ', '0', '1', '2', '3', '4', '5')
7. layout **=** [[psg.Text('Введите коэффициент при понижении температуры (0, 1)', font**=**'Default 12'), psg.InputText(key**=**'alpha', justification**=**'r')]] **+** \
8. [[psg.Text('Введите веса в матрицу смежности', font**=**'Default 12')]] **+** \
9. [[psg.Text(s, key**=**s, enable\_events**=**True, font**=**'Courier 14', size**=**(5, 1)) **for** i, s **in** enumerate(COL\_HEADINGS)]] **+** \
10. [[psg.T(r, size**=**(4, 1))] **+** [psg.Input(randint(0, 10), justification**=**'r', key**=**(r, c), disabled**=**False) **for** c **in** range(MAX\_COLS)] **for** r **in** range(MAX\_ROWS)] **+** \
11. [[psg.Button('Рассчитать')]] **+** \
12. [[psg.Text('Исходный граф                                                  Кратчайший гамильтонов цикл',
13. font**=**'Default 12', visible**=**False, key**=**'t1')]] **+** \
14. [[psg.Image('pictures/initial.png', key**=**'init\_img', visible**=**False, size**=**(10, 10))] **+**
15. [psg.Image('pictures/0.png', key**=**'i\_img', visible**=**False, size**=**(10, 10))]] **+** \
16. [[psg.Text('Кратчайший гамильтонов цикл', font**=**'Default 12', visible**=**False, key**=**'t2')]] **+** \
17. [[psg.Button('Назад', visible**=**False, key**=**'back\_but'), psg.Button('Далее', visible**=**False, key**=**'next\_but')]]
19. n **=** 0
20. data **=** []
21. max\_n **=** len(data) **-** 1
22. alpha **=** **-**1.0
24. window **=** psg.Window('Метод имитации отжига', layout, size**=**(900, 600), default\_element\_size**=**(8, 1), element\_padding**=**(1, 1), return\_keyboard\_events**=**True)
25. **while** True:
26. event, values **=** window.read()
27. print(event, values)
28. **if** event **in** (None, 'Exit'):
29. **break**
31. **if** event **==** 'Рассчитать':
32. # по диагонали ставим нули и запрещаем пользователю ввод
33. **for** i **in** range(MAX\_ROWS):
34. **for** j **in** range(MAX\_COLS):
35. **if** i **==** j:
36. window[(i, j)].update(0, disabled**=**True)
38. # считываем таблицу смежности с GUI
39. graph **=** [[int(values[(row, col)]) **for** col **in** range(MAX\_COLS)] **for** row **in** range(MAX\_ROWS)]
40. alpha **=** float(values['alpha'])
42. # запускаем алгоритм
43. data **=** algorithm.simulated\_annealing\_method(graph, alpha)
44. max\_n **=** len(data) **-** 1
46. # отображаем результаты
47. window['t1'].update(f'Исходный граф                                                                    Текущий гамильтонов цикл',
48. visible**=**True)
49. window['init\_img'].update('pictures/initial.png', visible**=**True)
50. window['i\_img'].update('pictures/0.png', visible**=**True)
51. **if** max\_n > 0:
52. window['back\_but'].update(visible**=**True, disabled**=**True)
53. window['next\_but'].update(visible**=**True, disabled**=**False)
55. # логика кнопки "Назад"
56. **if** event **==** 'back\_but':
57. n **-=** 1
58. window['i\_img'].update(f'pictures/{n}.png')
59. window['next\_but'].update(disabled**=**False)
60. **if** n **==** 0:
61. window['back\_but'].update(disabled**=**True)
63. # логика кнопки "Далее"
64. **if** event **==** 'next\_but':
65. n **+=** 1
66. window['i\_img'].update(f'pictures/{n}.png')
67. window['back\_but'].update(disabled**=**False)
68. **if** n **==** max\_n:
69. window['next\_but'].update(disabled**=**True)
71. window.close()