**Санкт-Петербургский государственный университет**

**Факультет прикладной математики и процессов управления**

**Лабораторная работа №4**

**Решение задачи о коммивояжере с помощью метода ближайшего соседа**

|  |  |  |
| --- | --- | --- |
| 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 ближайшего соседа для решения задачи о коммивояжере на примере графов.

1. **Задачи**

* Изучить метод ближайшего соседа
* Написать программу поиска кратчайшего гамильтонова цикла в графе с помощью метода ближайшего соседа
* Протестировать программу
* Реализовать GUI
* Найти кратчайший гамильтонов цикл для заданного графа

1. **Теоритическая часть**

Среди методов решения задачи коммивояжёра метод ближайшего соседа привлекает простотой алгоритма. Метод ближайшего соседа 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. Поставить все вершины как не посещённые.
2. Выбрать начальную вершину v и пометить её, как посещённую.
3. Выбрать наиближайшую не посещённую смежную вершину u к вершине v.
4. Поставить u как текущую вершину и пометить как посещённую.
5. Если все вершины посещены, то завершить алгоритм. Иначе, вернуться к шагу 3.

На выходе будем иметь последовательность вершин, предположительно оптимального решения.

1. **Описание программы**

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

* 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 |

* 1. **Описание функций и классов**

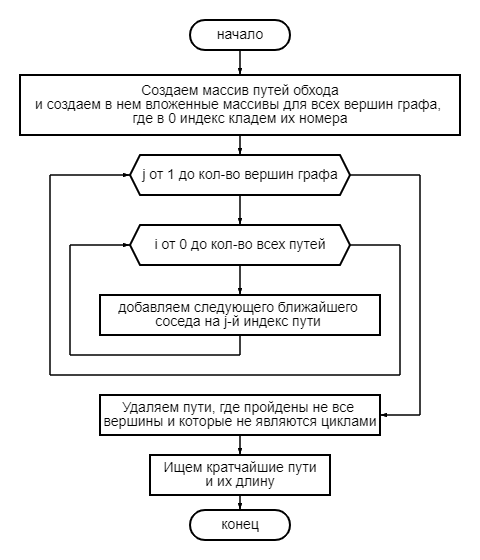
В таблице 2 представлено описание функций файла algorithm.py.

*Таблица 2. Функции algorithm.py*

|  |  |  |
| --- | --- | --- |
| **Имя функции** | **Входные данные** | **Описание функции** |
| find\_smallest\_el\_in\_row() | row – массив | Находит значения минимальных элементов в массиве и список их индексов |
| create\_start\_paths() | graph – матрица смежности графа | Создает массив путей обхода и создает вложенные массивы-пути обхода с началом в каждой вершине графа |
| add\_next\_neighbor() | path – путь, в который добавляется следующий элемент  ind – индекс шага пути  all\_paths – массив со всеми путями обхода  graph – матрица смежности графа | Для пути, переданного в функцию добавляет ближайшего соседа, которого еще нет в пути и, если их несколько, создает в all\_paths дополнительные ветки путей для всех ближайших соседей |
| zero\_v\_vert() | path – путь обхода  graph – матрица смежности графа | В исходной матрице смежности обнуляет значения всех столбцов (вершины, в которые можно пройти), которые есть в пути обхода, переданном в функцию |
| 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 |
| closest\_neighbor\_method() | graph – матрица смежности графа | Реализует работу метода ближайшего соседа на ориентированном взвешенном графе |

* 1. **Общий ход программы**

На рисунке 1 изображен ход подпрограммы, реализующей работу метода ближайшего соседа.



*Рисунок 1. Блок-схема подпрограммы, реализующей работу метода ближайшего соседа*

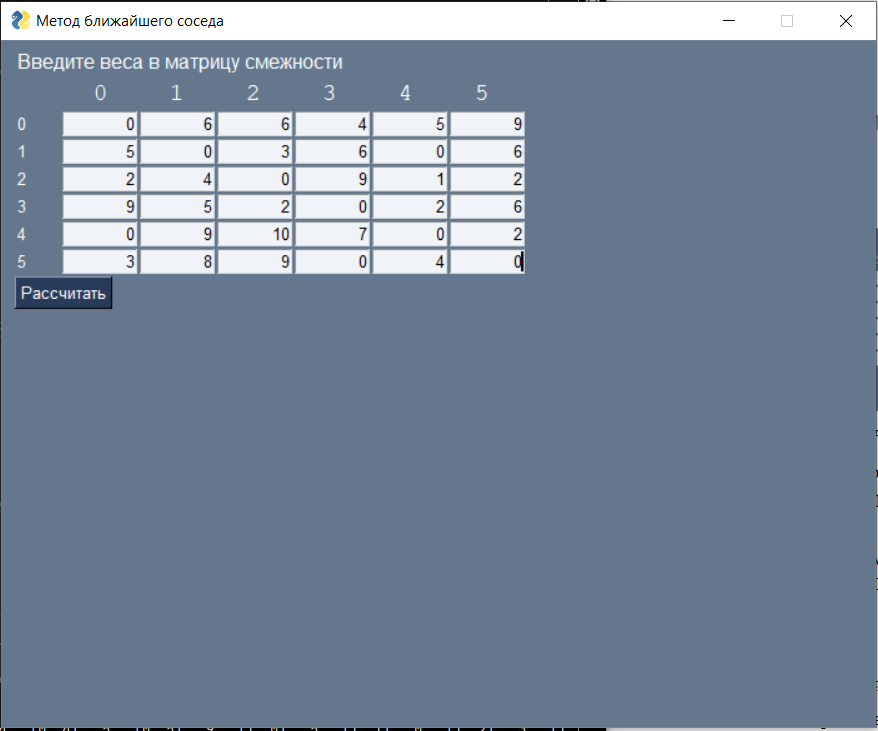
1. **Рекомендации программиста**

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**

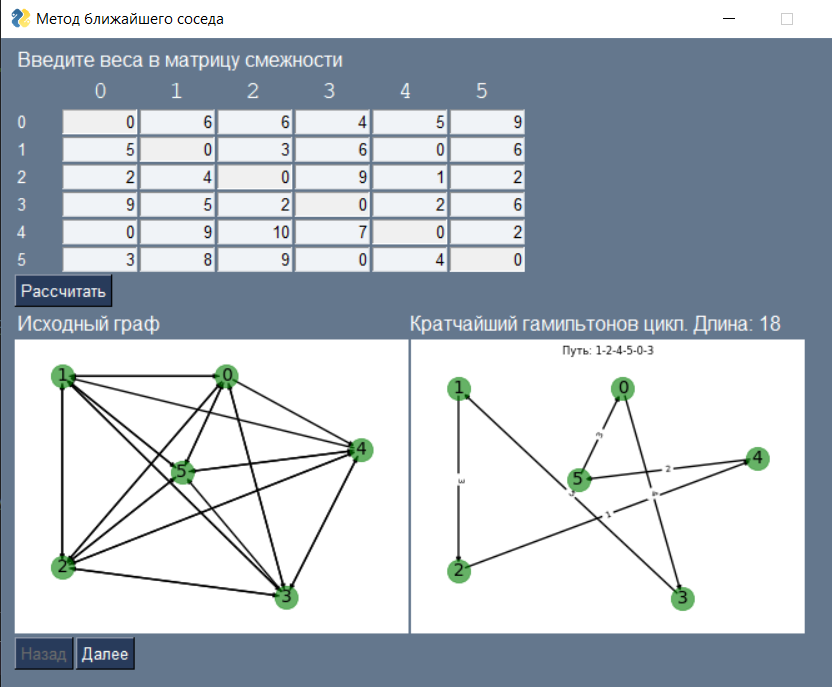
Запустите программу. Перед вами откроется начальное окно программы (Рис. 2).Введите веса в матрицу смежности графа, для которого хотите произвести расчеты или оставьте автоматически сгенерированные значения. Затем нажмите кнопку «Рассчитать» расчета наикратчайших гамильтоновых циклов, содержащихся в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



*Рисунок 2. Начальное окно*

По окончании расчетов будут выведены изображения графа, построенного на основе матрицы смежности и графы найденных кратчайших гамильтоновых циклов.

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

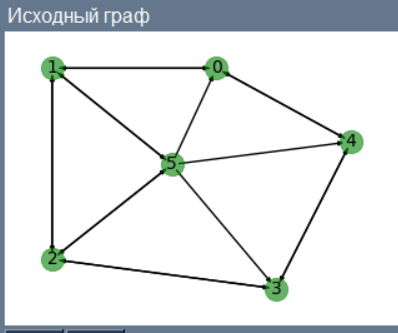


*Рисунок 3. Окно программы после нажатия кнопки «Рассчитать»*

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

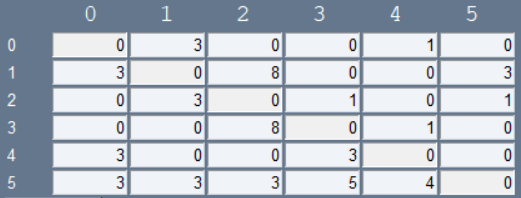
1. **Результаты работы алгоритма на заданном графе**

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



*Рисунок 4. Заданный орграф*

Матрица смежности имеет следующий вид (таблица 3).



*Рисунок 4. Матрица смежности*

По результатам работы алгоритма длина кратчайшего гамильтонового цикла равна 14. Пути, которые удовлетворяют этому условию:

[2, 3, 4, 0, 1, 5]

[3, 4, 0, 1, 5, 2]

[4, 0, 1, 5, 2, 3]

[5, 2, 3, 4, 0, 1]

[1, 5, 2, 3, 4, 0]

1. **Вывод**

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

**Приложения**

Приложение А. 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 веса в матрицу смежности', font**=**'Default 12')]] **+** \
8. [[psg.Text(s, key**=**s, enable\_events**=**True, font**=**'Courier 14', size**=**(5, 1)) **for** i, s **in** enumerate(COL\_HEADINGS)]] **+** \
9. [[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)] **+** \
10. [[psg.Button('Рассчитать')]] **+** \
11. [[psg.Text('Исходный граф                                                  Кратчайший гамильтонов цикл',
12. font**=**'Default 12', visible**=**False, key**=**'t1')]] **+** \
13. [[psg.Image('pictures/initial.png', key**=**'init\_img', visible**=**False, size**=**(10, 10))] **+**
14. [psg.Image('pictures/0.png', key**=**'i\_img', visible**=**False, size**=**(10, 10))]] **+** \
15. [[psg.Text('Кратчайший гамильтонов цикл', font**=**'Default 12', visible**=**False, key**=**'t2')]] **+** \
16. [[psg.Button('Назад', visible**=**False, key**=**'back\_but'), psg.Button('Далее', visible**=**False, key**=**'next\_but')]]
18. n **=** 0
19. best\_paths **=** []
20. best\_len **=** 0
21. max\_n **=** len(best\_paths) **-** 1
23. window **=** psg.Window('Метод ближайшего соседа', layout, size**=**(700, 550), default\_element\_size**=**(8, 1), element\_padding**=**(1, 1), return\_keyboard\_events**=**True)
24. **while** True:
25. event, values **=** window.read()
26. print(event, values)
27. **if** event **in** (None, 'Exit'):
28. **break**
30. **if** event **==** 'Рассчитать':
31. # по диагонали ставим нули и запрещаем пользователю ввод
32. **for** i **in** range(MAX\_ROWS):
33. **for** j **in** range(MAX\_COLS):
34. **if** i **==** j:
35. window[(i, j)].update(0, disabled**=**True)
37. # считываем таблицу смежности с GUI
38. graph **=** [[int(values[(row, col)]) **for** col **in** range(MAX\_COLS)] **for** row **in** range(MAX\_ROWS)]
40. # запускаем алгоритм
41. best\_len, best\_paths **=** algorithm.closest\_neighbor\_method(graph)
42. max\_n **=** len(best\_paths) **-** 1
44. # отображаем результаты
45. window['t1'].update(f'Исходный граф                                                  Кратчайший гамильтонов цикл. Длина: {best\_len}',
46. visible**=**True)
47. window['init\_img'].update('pictures/initial.png', visible**=**True)
48. #window['t2'].update(f'Кратчайший гамильтонов цикл. Длина: {best\_len}', visible=True)
49. window['i\_img'].update('pictures/0.png', visible**=**True)
50. **if** max\_n > 0:
51. window['back\_but'].update(visible**=**True, disabled**=**True)
52. window['next\_but'].update(visible**=**True, disabled**=**False)
54. # логика кнопки "Назад"
55. **if** event **==** 'back\_but':
56. n **-=** 1
57. window['i\_img'].update(f'pictures/{n}.png')
58. window['next\_but'].update(disabled**=**False)
59. **if** n **==** 0:
60. window['back\_but'].update(disabled**=**True)
62. # логика кнопки "Далее"
63. **if** event **==** 'next\_but':
64. n **+=** 1
65. window['i\_img'].update(f'pictures/{n}.png')
66. window['back\_but'].update(disabled**=**False)
67. **if** n **==** max\_n:
68. window['next\_but'].update(disabled**=**True)
70. window.close()

Приложение Б. algorithm.py

1. **import** matplotlib.pyplot as plt
2. **import** networkx as nx

5. **def** find\_smallest\_el\_in\_row(row):
6. """Находит значения минимальных элементов в массиве и список их индексов"""
7. min\_el **=** 1000
8. list **=** []
9. **for** i **in** range(len(row)):
10. **if** min\_el > row[i] !**=** 0:
11. min\_el **=** row[i]
12. list **=** [i]
13. **elif** row[i] **==** min\_el:
14. list.append(i)
15. **return** min\_el, list

18. **def** create\_start\_paths(graph):
19. """Cоздает массив со всеми возможными началами путей обхода"""
20. all\_paths **=** []
21. **for** i **in** range(len(graph)):
22. all\_paths.append([0, [i]])
23. **return** all\_paths

26. **def** add\_next\_neighbor(path, ind, all\_paths, graph):
27. """Добавляет все возможные ветки путей"""
28. new\_g **=** zero\_v\_vert(path, graph)
29. min\_s, list\_min\_s **=** find\_smallest\_el\_in\_row(new\_g[path[1][ind **-** 1]])
31. save\_path **=** path[1][:]
32. **if** len(list\_min\_s) !**=** 0:
33. path[0] **+=** min\_s
34. path[1].append(list\_min\_s[0])
36. **if** len(list\_min\_s) > 1:
37. **for** i **in** range(1, len(list\_min\_s)):
38. a **=** save\_path[:]
39. a.append(list\_min\_s[i])
40. all\_paths.append([path[0], a])

43. **def** zero\_v\_vert(path, graph):
44. """Обнуляет вертикальные столбцы в матрице graph по индексам из path"""
45. leng **=** len(graph[0])
46. **for** i **in** range(leng):
47. **if** i **in** path[1]:
48. **for** j **in** range(leng):
49. graph[j][i] **=** 0
50. **return** graph

53. **def** create\_multigraph\_struct(graph):
54. G **=** nx.MultiDiGraph()
55. **for** i **in** range(len(graph)):
56. **for** j **in** range(len(graph[0])):
57. **if** graph[i][j] !**=** 0 **and** i !**=** j:
58. G.add\_edge(f'{i}', f'{j}', weight**=**graph[i][j])
59. **return** G

62. **def** create\_graph\_struct(path\_arr, graph):
63. G **=** nx.DiGraph()
64. data **=** []
65. **for** j **in** range(len(path\_arr)):
66. **if** j **+** 1 < len(path\_arr):
67. data.append(path\_arr[j **+** 1])
68. data.append(path\_arr[0])
69. **for** i **in** range(len(path\_arr)):
70. G.add\_edge(f'{path\_arr[i]}', f'{data[i]}', weight**=**graph[path\_arr[i]][data[i]])
71. **return** G

74. **def** make\_plt(flag, path\_arr, graph, i):
75. pos **=** {'0': [0, 0.25],
76. '1': [**-**0.55, 0.25],
77. '2': [**-**0.55, **-**0.4],
78. '3': [0.2, **-**0.5],
79. '4': [0.45, 0],
80. '5': [**-**0.15, **-**0.0775903]}
81. path **=** ''
83. **if** flag **==** 'multi':
84. G **=** create\_multigraph\_struct(graph)
85. snapshot\_name **=** "pictures/initial.png"
86. **else**:
87. G **=** create\_graph\_struct(path\_arr, graph)
88. snapshot\_name **=** f"pictures/{i}.png"
89. path **=** f'{str(path\_arr[0])}-{str(path\_arr[1])}-' \
90. f'{str(path\_arr[2])}-{str(path\_arr[3])}-' \
91. f'{str(path\_arr[4])}-{str(path\_arr[5])}'
93. # nodes
94. nx.draw\_networkx\_nodes(G, pos, node\_size**=**700, node\_color**=**'green', alpha**=**0.6)
95. # edges
96. nx.draw\_networkx\_edges(G, pos, width**=**2)
97. # node labels
98. nx.draw\_networkx\_labels(G, pos, font\_size**=**20, font\_family**=**"sans-serif")
99. **if** flag **==** "basic":
100. # edge weight labels
101. edge\_labels **=** nx.get\_edge\_attributes(G, "weight")
102. nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels)
103. plt.title(f'Путь: {path}')
104. ax **=** plt.gca()
105. ax.margins(0.08)
106. plt.axis("off")
107. plt.tight\_layout()
109. plt.savefig(snapshot\_name, dpi**=**50, bbox\_inches**=**'tight')
110. plt.close()

113. **def** draw\_graph(paths\_arr, graph):
114. """Визуализация графов"""
116. flag **=** 'multi'
117. make\_plt(flag, paths\_arr[0], graph, 0)
119. # рисуем графы наикратчайших гамильтоновых циклов
120. **for** i **in** range(len(paths\_arr)):
121. flag **=** 'basic'
122. make\_plt(flag, paths\_arr[i], graph, i)

125. **def** closest\_neighbor\_method(graph):
126. # создаем массив со всеми возможными началами путей обхода
127. all\_paths **=** create\_start\_paths(graph)
129. # каждый путь продолжаем до конца
130. **for** j **in** range(1, len(graph)):
131. **for** i **in** range(len(all\_paths)):
132. deep\_graph **=** [x[:] **for** x **in** graph]
133. add\_next\_neighbor(all\_paths[i], j, all\_paths, deep\_graph)
135. # удаляем пути, где пройдены не все вершины, которые не явл циклами и ищем кратчайшие
136. best\_paths **=** []
137. best\_len **=** 1000
138. mas\_del **=** []
139. **for** i **in** range(len(all\_paths)):
140. **if** len(all\_paths[i][1]) **==** len(graph) **and** \
141. graph[all\_paths[i][1][len(graph) **-** 1]][all\_paths[i][1][0]] !**=** 0:
142. print(i)
143. all\_paths[i][0] **+=** graph[all\_paths[i][1][len(graph) **-** 1]][all\_paths[i][1][0]]
144. **if** all\_paths[i][0] < best\_len:
145. best\_len **=** all\_paths[i][0]
146. **else**:
147. mas\_del.append(all\_paths[i])
149. **for** l **in** mas\_del:
150. all\_paths.remove(l)
152. print('all paths')
153. **for** el **in** all\_paths:
154. print(el)
155. **if** el[0] **==** best\_len:
156. best\_paths.append(el[1])
158. print('best paths')
159. print('len = ', best\_len)
160. **for** el **in** best\_paths:
161. print(el)
163. draw\_graph(best\_paths, graph)
165. **return** best\_len, best\_paths