

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №3**  
**по дисциплине «Построение и анализ алгоритмов»**  
**ТЕМА: АЛГОРИТМ КНУТА-МОРРИСА-ПРАТТА**

Студент гр. 1384

Усачева Д.В.

Преподаватель

Шевелева А. М.

Санкт-Петербург

2023

### **Цель работы.**

Изучить возможные реализации построения минимального гамильтонова цикла – решения задачи о коммивояжере, а также их оптимизации.

### **Задание.**

Дана карта городов в виде ассиметричного, неполного графа  $G = (V, E)$ , где  $V(|V|=n)$  – это вершины графа, соответствующие городам;  $E(|E|=m)$  – это ребра между вершинами графа, соответствующие путям сообщения между этими городами.

Каждому ребру  $m_{ij}$  (переезд из города  $i$  в город  $j$ ) можно сопоставить критерий выгодности маршрута (вес ребра) равный  $w_i$  (натуральное число  $[1, 1000]$ ),  $m_{ij} = \text{inf}$ , если  $i=j$ .

Если маршрут включает в себя ребро  $m_{ij}$ , то  $x_{ij} = 1$ , иначе  $x_{ij} = 0$ .

Требуется найти минимальный маршрут (минимальный гамильтонов цикл):

Входные параметры:

Матрица графа из текстового файла.

inf 1 2 2

- inf 1 2

- 1 inf 1

1 1 - inf

Выходные параметры:

Кратчайший путь, вес кратчайшего пути, скорость решения задачи.

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

// Задача должна решаться на размере матрицы 20x20 не дольше 3 минут в среднем.

### **Выполнение работы.**

Данный код решает задачу коммивояжера при помощи алгоритма Беллмана-Хелда-Карпа.

Алгоритм Беллмана-Хелда-Карпа - это алгоритм поиска кратчайшего пути в графе, позволяющий обойти заданные вершины по определённому маршруту единожды. Суть алгоритма заключается в последовательном рассмотрении всех возможных маршрутов, проходящих через каждую из вершин в заданном порядке.

Класс TSP (Traveling Salesman Problem) содержит функции:

- `init` – конструктор класса для инициализации переменных
- `input_graph` – метод для ввода графа из файла и создания начальных параметров для `min_way_table`, `set_of_values`, `key_addition`
- `min_set` – метод для поиска минимального пути из подмножеств к вершинам
- `find_min_hamiltonian_path` – метод для поиска минимального гамильтонова пути

Он принимает на вход файл с графом, который описывает матрицу расстояний между каждой парой вершин в графе. Файл должен содержать матрицу в виде квадратной таблицы, где каждый элемент матрицы - это расстояние между соответствующими вершинами.

Далее происходит заполнение массива `matrix_graph` из графа файла, если между вершинами нет прямого пути, то записывается значение ``math.inf`` – бесконечность.

Затем создаются начальные значения для `min_path_table`, `set_of_values`, и `key_addition`. Значения `min_path_table` — это словарь, который хранит минимальные пути для текущего шага, где ключ — это множество, из которого ведется поиск минимального пути к вершинам. `set_of_values` представляет собой множество ключей графа для текущего шага. `key_addition` — массив дополнений для множества ключей.

Для поиска минимального пути из подмножества к вершинам используется метод `min_set`. В ней подмножество разбивается на подмножества меньшей длины и дополняющие их вершины. Затем выбирается

минимальный путь к каждой из целевых вершин из всех разбитых на части подмножеств с помощью таблицы из предыдущего шага.

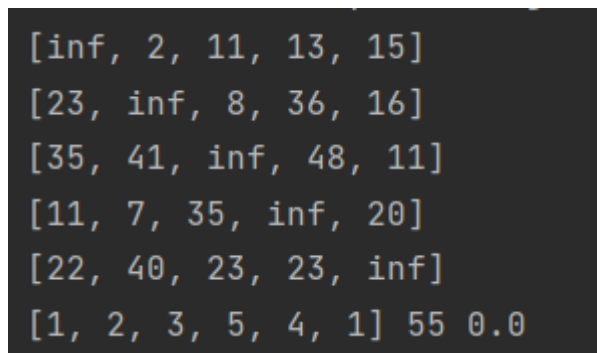
Для поиска минимального гамильтонова цикла используется метод `find_min_hamiltonian_path`. Его поиск разделен на шаги, номер шага соответствует длине рассматриваемого на этом шаге множества. Сначала мы сохраняем данные таблицы путей и массива ее ключей из предыдущего шага, далее заполняем значения массива ключей для текущего шага. После этого идет заполнение таблицы для текущего шага при помощи функции `min_set`. На каждом шаге хранится информация только о текущей и предыдущей итерации, остальные данные не являются нужными, поэтому удаляются. Также в методе рассматриваются случаи матрицы размера 1 и отсутствия прямого гамильтонова цикла.

В итоге, получается минимальная замкнутая гамильтонова цепь в графе, проходящая через каждую вершину только один раз.

### **Тестирование.**

Результаты тестирования представлены на рисунках 1 — 5.

1. Проверка работы программы на произвольном полном графе на 5-и вершинах (см. рисунок 1)



```
[inf, 2, 11, 13, 15]
[23, inf, 8, 36, 16]
[35, 41, inf, 48, 11]
[11, 7, 35, inf, 20]
[22, 40, 23, 23, inf]
[1, 2, 3, 5, 4, 1] 55 0.0
```

Рисунок 1 – Тестирование матрицы 5x5

2. Проверка работы программы на произвольном полном графе на 20-и вершинах (см. рисунок 2)

```
[inf, 690, 470, 814, 71, 255, 333, 664, 508, 656, 613, 886, 334, 842, 967, 754, 282, 781, 911, 781]
[225, inf, 614, 571, 761, 613, 42, 894, 636, 769, 267, 421, 541, 748, 472, 458, 210, 616, 993, 779]
[362, 503, inf, 161, 10, 864, 262, 729, 200, 770, 22, 404, 252, 723, 379, 282, 308, 792, 862, 862]
[500, 947, 630, inf, 300, 446, 192, 654, 856, 858, 103, 537, 950, 255, 352, 908, 788, 604, 306, 460]
[512, 372, 535, 641, inf, 715, 931, 67, 350, 18, 761, 416, 443, 83, 912, 840, 764, 160, 737, 986]
[314, 175, 354, 281, 209, inf, 433, 330, 83, 680, 33, 803, 950, 278, 178, 130, 841, 478, 364, 15]
[649, 105, 240, 902, 388, 726, inf, 725, 53, 890, 585, 221, 654, 28, 54, 243, 545, 97, 907, 657]
[396, 143, 577, 521, 263, 74, 267, inf, 225, 417, 494, 194, 201, 744, 380, 194, 305, 971, 994, 916]
[635, 257, 269, 846, 65, 733, 334, 809, inf, 584, 204, 32, 707, 281, 892, 983, 158, 869, 275, 830]
[144, 335, 490, 276, 667, 428, 311, 881, 576, inf, 172, 131, 336, 304, 986, 48, 97, 824, 729, 822]
[729, 70, 315, 429, 843, 38, 763, 110, 125, 331, inf, 756, 64, 382, 665, 567, 108, 407, 783, 26]
[996, 321, 150, 804, 105, 615, 609, 57, 351, 190, 96, inf, 649, 995, 814, 437, 530, 473, 578, 567]
[87, 749, 96, 579, 564, 942, 531, 710, 454, 785, 155, 259, inf, 829, 527, 353, 885, 519, 2, 721]
[879, 871, 399, 692, 953, 370, 537, 197, 35, 584, 471, 158, 798, inf, 275, 418, 878, 684, 542, 375]
[227, 971, 142, 455, 209, 903, 758, 657, 254, 547, 289, 348, 359, 81, inf, 830, 108, 243, 30, 873]
[471, 644, 366, 861, 561, 837, 615, 581, 620, 338, 346, 266, 123, 454, 819, inf, 97, 937, 684, 924]
[60, 765, 95, 801, 975, 338, 230, 49, 170, 467, 156, 703, 261, 744, 552, 860, inf, 458, 195, 501]
[703, 190, 457, 151, 316, 585, 192, 132, 837, 14, 759, 412, 314, 127, 998, 325, 158, inf, 277, 87]
[728, 47, 264, 507, 646, 548, 364, 327, 655, 243, 943, 430, 817, 309, 836, 443, 259, 270, inf, 626]
[838, 394, 748, 425, 697, 171, 223, 124, 2, 869, 298, 205, 639, 444, 330, 108, 371, 618, 671, inf]
[1, 5, 14, 9, 12, 8, 6, 20, 16, 17, 3, 4, 11, 13, 19, 2, 7, 15, 18, 10, 1] 1541 83.40079021453857
```

Рисунок 2 – Тестирование матрицы 20x20

3. Проверка работы программы на произвольном полном графе на 20-и вершинах (см. рисунок 3)

```
[inf, 851, 234, 496, 30, 533, 151, 61, 280, 299, 986, 112, 352, 530, 467, 114, 341, 519, 295, 415]
[809, inf, 723, 301, 478, 120, 891, 652, 199, 60, 943, 133, 954, 387, 252, 975, 869, 746, 796, 967]
[503, 707, inf, 273, 143, 915, 55, 678, 632, 998, 694, 532, 102, 63, 113, 105, 749, 984, 587, 462]
[863, 711, 760, inf, 661, 310, 12, 334, 467, 567, 718, 272, 108, 364, 627, 156, 70, 631, 170, 543]
[815, 988, 86, 130, inf, 61, 803, 640, 455, 741, 200, 495, 633, 716, 257, 935, 567, 465, 724, 440]
[63, 14, 685, 869, 84, inf, 171, 486, 203, 635, 76, 17, 443, 426, 804, 675, 176, 927, 961, 72]
[175, 939, 874, 26, 147, 368, inf, 341, 105, 905, 683, 190, 923, 831, 972, 223, 977, 577, 997, 812]
[833, 126, 395, 766, 983, 727, 584, inf, 130, 289, 514, 961, 146, 223, 469, 171, 49, 135, 352, 209]
[23, 743, 736, 946, 927, 961, 21, 909, inf, 703, 178, 943, 902, 788, 829, 647, 969, 320, 582, 125]
[316, 323, 67, 816, 523, 356, 677, 912, 437, inf, 327, 116, 239, 216, 657, 400, 631, 220, 601, 566]
[502, 324, 709, 732, 556, 2, 711, 332, 895, 625, inf, 953, 553, 775, 771, 230, 461, 463, 733, 627]
[595, 690, 141, 518, 350, 199, 693, 301, 583, 88, 813, inf, 967, 337, 508, 716, 923, 601, 331, 677]
[529, 389, 960, 735, 837, 377, 498, 410, 851, 718, 591, 912, inf, 139, 325, 494, 915, 26, 37, 96]
[145, 843, 641, 634, 449, 777, 349, 449, 736, 689, 14, 208, 680, inf, 948, 907, 48, 560, 676, 284]
[180, 513, 974, 187, 823, 911, 888, 28, 208, 422, 851, 19, 774, 132, inf, 522, 498, 177, 275, 305]
[421, 843, 638, 609, 25, 632, 59, 446, 669, 138, 501, 261, 357, 424, 594, inf, 404, 348, 788, 21]
[689, 318, 511, 587, 836, 847, 447, 834, 41, 471, 205, 180, 966, 990, 187, 510, inf, 616, 698, 967]
[775, 776, 580, 481, 974, 852, 210, 303, 965, 851, 344, 302, 995, 634, 857, 900, 956, inf, 11, 676]
[671, 394, 740, 914, 389, 148, 147, 8, 239, 62, 655, 339, 674, 751, 505, 279, 53, 272, inf, 249]
[818, 834, 334, 105, 386, 997, 807, 200, 194, 132, 82, 3, 34, 92, 175, 212, 465, 155, 934, inf]
[1, 5, 15, 14, 11, 6, 2, 12, 10, 3, 7, 4, 16, 20, 13, 18, 19, 8, 17, 9, 1] 1187 81.42156267166138
```

Рисунок 3 – Тестирование матрицы 20x20

4. Проверка работы программы на слабо связном графе без гамильтонова цикла (см. рисунок 4).

```
[inf, 1, inf, 1, 1, inf]
[1, inf, 1, inf, 1, inf]
[inf, inf, inf, 1, 1, inf]
[inf, inf, 1, inf, 1, inf]
[inf, inf, inf, 1, inf, 1]
[inf, inf, 1, inf, 1, inf]
No way 0.0
```

Рисунок 4 – Тестирование на слабо связном графе без гамильтонова цикла

5. Проверка работы программы на графе с одинаковым весом всех, кроме одного, ребер (см. рисунок 5).

```
[inf, 2, 2, 2, 2, 2]
[2, inf, 2, 2, 2, 2]
[2, 2, inf, 2, 2, 2]
[1, 2, 2, inf, 2, 2]
[2, 2, 2, 2, inf, 2]
[2, 2, 2, 2, 2, inf]
[1, 6, 5, 3, 2, 4, 1] 11 0.000995635986328125
```

Рисунок 5 – Тестирование матрицы с одинаковым весом ребер, кроме одного

### **Выводы.**

В рамках данной лабораторной работы был изучен алгоритм, который решает задачу коммивояжера - поиск минимального гамильтонова цикла в графе. Полный перебор работает очень долго, метод ветвей и границ не стабилен, поэтому в качестве алгоритма для решения задачи коммивояжера был выбран алгоритм Беллмана-Хелда-Карпа.

В качестве решения поставленной задачи была написана программа, которая с помощью алгоритма Беллмана-Хелда-Карпа решает задачу коммивояжера для графов размера 20 на 20 менее, чем за две минуты.

Были рассмотрены особые условия (граф несвязен, все ребра кроме 1 равны по весу и т.д.), и при всех таких условиях программа работала корректно.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: lb3.py

```
import math
import time

class TSP:
    '''
    Коструктор класса
    matrix_graph - граф, введенный из файла
    key_addition - массив дополнений для множества ключей, состоит из
чисел от 2 до count (тип значений - tuple)
    min_path_table - словарь, для хранения минимальных путей для текущего
шага, где ключ это множество, из которого
    ведется поиск минимального пути к вершинам, значение - массив кортежей
длины 3 (вершина
    в которую идем, минимальный путь к этой вершине, длина этого пути)
    previous_min_path_table - словарь, для хранения таблицы для
пердыдущего шага
    set_of_values - множество ключей графа, для текущего шага
    count - количество строк/столбцов графа
    '''
    def __init__(self):
        self.matrix_graph = []
        self.key_addition = []
        self.min_path_table = dict()
        self.previous_min_path_table = dict()
        self.set_of_values = []
        self.count = 0

    '''
    Функция для ввода графика из файла, также в ней задаются начальные
парметры для
    min_way_table, set_of_values, key_addition
    '''
    def input_graph(self, file_name):
        file = open(file_name)
        for line in file:
            line_matrix = []
            for i in line.split():
```

```

        if i == '-1' or i == 'inf' or i == '-' or int(i) >= 100000:
            line_matrix.append(math.inf)
        else:
            line_matrix.append(int(i))
        self.matrix_graph.append(line_matrix)
    file.close()
    self.count = len(self.matrix_graph[0])
    for i in range(2, self.count + 1):
        self.min_path_table[tuple([i])] = []
        for j in range(2, self.count + 1):
            if i != j:
                self.min_path_table[tuple([i])].append(
                    (j, [1, i], self.matrix_graph[0][i - 1] +
self.matrix_graph[i - 1][j - 1]))
                self.set_of_values.append(tuple([i]))
        self.key_addition = self.set_of_values.copy()

'''
Функция для поиска минимального пути из подмножества к вершинам
'''
def min_set(self, set):
    subsets = []
    complementary_vertex = []
    final_vertex = []
    min_path_set = []
    set = list(set)
    for i in range(2, self.count + 1):
        if i in set:
            complementary_vertex.append(i)
            new_set = set.copy()
            new_set.remove(i)
            subsets.append(tuple(new_set))
        else:
            final_vertex.append(i)
    if not final_vertex:
        final_vertex.append(1)
    for v in final_vertex:
        path_length = []
        path = []
        for i in range(len(subsets)):
            for j in self.previous_min_path_table[subsets[i]]:
                if j[0] == complementary_vertex[i]:

```



```

        path_length.append(j[2]
self.matrix_graph[complementary_vertex[i] - 1][v - 1])
        path.append(j)
        best_id = path_length.index(min(path_length))
        min_way = path[best_id][1].copy()
        min_way.append(path[best_id][0])
        min_path_set.append((v, min_way, min(path_length)))
    return min_path_set

'''
Функция для поиска минимального гамильтонова пути
'''
def find_min_hamiltonian_path(self):
    if self.count == 1:
        print("[1, 1]", self.matrix_graph[0][0])
        return 0
    start = time.time()
    for step in range(2, self.count):
        previous_set_of_values = self.set_of_values.copy()
        self.previous_min_path_table = self.min_path_table.copy()
        self.min_path_table.clear()
        self.set_of_values.clear()
        for prev_set in previous_set_of_values:
            for key_addition in self.key_addition:
                current_set = set(prev_set + key_addition)
                current_set = list(current_set)
                current_set.sort()
                current_set = tuple(current_set)
                if len(current_set) == step:
                    self.set_of_values.append(current_set)
        self.set_of_values = set(self.set_of_values)
        self.set_of_values = list(self.set_of_values)
        for i in self.set_of_values:
            self.min_path_table[i] = self.min_set(i)
    end = time.time() - start
    if self.min_path_table[self.set_of_values[0]][0][2] == math.inf:
        print("No way", end)
    else:
        self.min_path_table[self.set_of_values[0]][0][1].append(1)
        print(self.min_path_table[self.set_of_values[0]][0][1],
self.min_path_table[self.set_of_values[0]][0][2],
        end)

```

```
ts = TSP()  
ts.input_graph('test1.txt')  
ts.find_min_hamiltonian_path()
```