Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
|  |
| Кафедра «Информатика» |
| кафедра |

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ**

|  |
| --- |
| Муравьиные алгоритмы. Задача коммивояжера |
| Тема |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  |  |  | Р. Ю. Царев |
|  | |  |  |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ19-17/1б 031939175 | | |  |  |  | А. Д. Непомнящий |
|  | номер группы, зачетной книжки | | |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2021

1. Цель работы

Изучение принципа решения задачи коммивояжера с помощью муравьиного алгоритма.

1. Задачи

Требуется разработать программу, которая с помощью муравьиного алгоритма решает задачу коммивояжера - обходит все вершины графа, при этом длина пути минимальная из возможных.

Предъявлены следующие требования к выполнению работы.

1. Строгое соответствие программы и результатов ее работы с полученным заданием.
2. Самостоятельные тестирование и отладка программы.
3. Предоставление демонстрационного примера и исходного текста программы для защиты.
4. Предоставление отчета по практическому заданию, содержащего описание реализованного алгоритма, программы, результатов работы программы (отчет необходимо загрузить на сайт курса).
5. Описание реализованного алгоритма

Реализовано муравьиный алгоритм по поиску гамильтоновой цепи минимального веса.

1. Описание программы

Для решения задачи был написан модуль на языке Python версии 3.9.

Поиск кратчайшего пути коммивояжера выполняет функция antTSP(). Для демонстрации программы было создано две матрицы весов графов с пятью и двадцатью пятью вершинами.

Для проведения простых тестов производительности алгоритма был использован пакет timethis. Для сравнения был так же взят алгоритм ближайшего соседа из пакета tspsolve.

Листинг 1 – Код в файле main.py

import random

import numpy as np

from ptimeit import timethis, Timer

from tspsolve import nearest\_neighbor

def get\_len\_t(D, T):

len\_t = D[T[0]][T[-1]]

for i in range(len(T) - 1):

len\_t += D[T[i]][T[i + 1]]

return len\_t

def desire(cur\_city, next\_city, tau, eta, alpha, beta):

return tau[cur\_city][next\_city] \*\* alpha \* eta[cur\_city][next\_city] \*\* beta

def choose\_next\_city(cur\_city, allow\_list, tau, eta, alpha, beta):

desires = [desire(cur\_city, i, tau, eta, alpha, beta) for i in

allow\_list]

sum\_desires = sum(desires)

probabilities = [val\_desire / sum\_desires for val\_desire in desires]

r = random.random()

p = 0

for i in range(len(probabilities)):

p += probabilities[i]

if r <= p:

return allow\_list[i]

def build\_route(cur\_city, allow\_list, tabu\_list, tau, eta, alpha, beta):

while allow\_list:

allow\_list.remove(cur\_city)

tabu\_list.append(cur\_city)

build\_route(

choose\_next\_city(cur\_city, allow\_list, tau, eta, alpha, beta),

allow\_list, tabu\_list, tau, eta, alpha, beta)

return tabu\_list

Продолжение листинга 1

def get\_routes(m, n, tau, eta, alpha, beta):

routes = [[] for \_ in range(m)]

for k in range(m):

allow\_list = [i for i in range(n)]

tabu\_list = []

routes[k] = build\_route(k, allow\_list, tabu\_list, tau, eta, alpha,

beta)

return routes

def update\_tau(D, tau, routes, Q, q):

n = len(D)

for i in range(n):

for j in range(n):

tau[i][j] = round(tau[i][j] \* q, 3)

for route in routes:

delta\_tau = Q / get\_len\_t(D, route)

for i in range(n - 1):

tau[route[i]][route[i + 1]] += round(delta\_tau, 3)

tau[route[i + 1]][route[i]] += round(delta\_tau, 3)

tau[route[0]][route[-1]] += round(delta\_tau, 3)

tau[route[-1]][route[0]] += round(delta\_tau, 3)

return tau

def antTSP(tmax, D, alpha, beta, Q, tau0, q):

n = len(D)

m = n

eta = [[0] \* len(D[i]) for i in range(n)]

tau = [[0] \* len(D[i]) for i in range(n)]

for i in range(n):

for j in range(n):

if i != j:

eta[i][j] = round(1 / D[i][j], 3)

tau[i][j] = tau0

else:

Продолжение листинга 1

tau[i][j] = 0

T\_ = [i for i in range(n)]

L\_ = get\_len\_t(D, T\_)

for i in range(tmax):

routes = get\_routes(m, n, tau, eta, alpha, beta)

for route in routes:

L\_k = get\_len\_t(D, route)

if L\_k < L\_:

T\_ = route

L\_ = L\_k

if i == (tmax - 1):

return T\_

else:

tau = update\_tau(D, tau, routes, Q, q)

def nearestNeighborTSP(D, label):

length = len(D)

vertices = []

H = []

s = np.random.randint(len(D))

v = s

while v not in vertices:

w = 10000003

index = -1

for i in range(length):

if i not in vertices and i != v and D[v][i] < w:

w = D[v][i]

index = i

if w == 10000003 or index == -1:

break

if len(H) == 0:

H.append(label[v])

H.append(label[index])

vertices.append(v)

v = index

H.pop()

Продолжение листинга 1

return H

@timethis(name="Ant tsp algorithm")

def benchmarkAntTSP():

antTSP(tmax=10, D=distances\_matrix,

alpha=1, beta=1, Q=10, tau0=2,

q=0.64)

return

@timethis(name="Nearest neighbor tsp algorithm")

def benchmarkNearestNeighbourTSP():

nearest\_neighbor(np.array(distances\_matrix))

return

if \_\_name\_\_ == "\_\_main\_\_":

#distances\_matrix = [

# [0, 156, 479, 630, 246, 420, 184, 267, 408, 617, 733, 913, 1070, 736,

# 625, 924, 1259, 462, 480, 345, 171, 376, 157, 328, 363],

# [156, 0, 323, 474, 402, 576, 340, 423, 564, 773, 889, 1069, 1226, 892,

# 781, 1080, 1415, 618, 636, 501, 327, 532, 313, 484, 519],

# [479, 323, 0, 151, 725, 696, 663, 746, 887, 1096, 1212, 1392, 1549,

# 1215, 1104, 1403, 1738, 941, 959, 824, 650, 855, 636, 807, 842],

# [630, 474, 151, 0, 719, 545, 814, 819, 1038, 1247, 1363, 1543, 1700,

# 1366, 1255, 1554, 1889, 1092, 1110, 975, 801, 1006, 787, 958, 993],

# [246, 402, 725, 719, 0, 174, 183, 100, 407, 616, 732, 912, 1069, 735,

# 624, 943, 1278, 708, 726, 591, 417, 375, 403, 574, 609],

# [420, 576, 696, 545, 174, 0, 357, 274, 581, 790, 906, 1086, 1243, 909,

# 798, 1117, 1452, 882, 900, 765, 591, 549, 577, 748, 783],

# [184, 340, 663, 814, 183, 357, 0, 83, 224, 433, 549, 729, 886, 552,

# 441, 760, 1095, 646, 664, 529, 355, 192, 341, 512, 547],

# [267, 423, 746, 819, 100, 274, 83, 0, 307, 516, 632, 812, 969, 635,

# 524, 843, 1178, 729, 747, 612, 438, 275, 424, 595, 630],

# [408, 564, 887, 1038, 407, 581, 224, 307, 0, 209, 325, 505, 662, 328,

# 217, 536, 871, 870, 888, 753, 579, 416, 565, 736, 771],

# [617, 773, 1096, 1247, 616, 790, 433, 516, 209, 0, 116, 296, 453, 537,

# 426, 745, 1080, 1079, 1097, 962, 788, 625, 774, 945, 980],

# [733, 889, 1212, 1363, 732, 906, 549, 632, 325, 116, 0, 180, 337, 431,

# 542, 639, 974, 1101, 1213, 1078, 904, 741, 890, 1061, 1096],

Продолжение листинга 1

# [913, 1069, 1392, 1543, 912, 1086, 729, 812, 505, 296, 180, 0, 157,

# 251, 362, 459, 794, 921, 1133, 1258, 1084, 633, 1070, 1241, 1276],

# [1070, 1226, 1549, 1700, 1069, 1243, 886, 969, 662, 453, 337, 157, 0,

# 342, 453, 550, 885, 1012, 1224, 1359, 1241, 724, 1227, 1398, 1433],

# [736, 892, 1215, 1366, 735, 909, 552, 635, 328, 537, 431, 251, 342, 0,

# 111, 208, 543, 670, 882, 1017, 907, 382, 893, 1064, 1099],

# [625, 781, 1104, 1255, 624, 798, 441, 524, 217, 426, 542, 362, 453,

# 111, 0, 319, 654, 781, 993, 970, 796, 493, 782, 953, 988],

# [924, 1080, 1403, 1554, 943, 1117, 760, 843, 536, 745, 639, 459, 550,

# 208, 319, 0, 335, 462, 674, 809, 983, 590, 1081, 1252, 1287],

# [1259, 1415, 1738, 1889, 1278, 1452, 1095, 1178, 871, 1080, 974, 794,

# 885, 543, 654, 335, 0, 797, 1009, 1144, 1318, 925, 1416, 1587, 1622],

# [462, 618, 941, 1092, 708, 882, 646, 729, 870, 1079, 1101, 921, 1012,

# 670, 781, 462, 797, 0, 212, 347, 521, 838, 619, 790, 825],

# [480, 636, 959, 1110, 726, 900, 664, 747, 888, 1097, 1213, 1133, 1224,

# 882, 993, 674, 1009, 212, 0, 135, 309, 856, 637, 808, 843],

# [345, 501, 824, 975, 591, 765, 529, 612, 753, 962, 1078, 1258, 1359,

# 1017, 970, 809, 1144, 347, 135, 0, 174, 721, 502, 673, 708],

# [171, 327, 650, 801, 417, 591, 355, 438, 579, 788, 904, 1084, 1241,

# 907, 796, 983, 1318, 521, 309, 174, 0, 547, 328, 499, 534],

# [376, 532, 855, 1006, 375, 549, 192, 275, 416, 625, 741, 633, 724, 382,

# 493, 590, 925, 838, 856, 721, 547, 0, 533, 704, 739],

# [157, 313, 636, 787, 403, 577, 341, 424, 565, 774, 890, 1070, 1227,

# 893, 782, 1081, 1416, 619, 637, 502, 328, 533, 0, 171, 520],

# [328, 484, 807, 958, 574, 748, 512, 595, 736, 945, 1061, 1241, 1398,

# 1064, 953, 1252, 1587, 790, 808, 673, 499, 704, 171, 0, 691],

# [363, 519, 842, 993, 609, 783, 547, 630, 771, 980, 1096, 1276, 1433,

# 1099, 988, 1287, 1622, 825, 843, 708, 534, 739, 520, 691, 0],

#]

distances\_matrix = [[0, 10, 2, 4, 14],

[10, 0, 6, 13, 8],

[2, 6, 0, 2, 7],

[4, 13, 2, 0, 20],

[14, 8, 7, 20, 0]]

antResult = antTSP(tmax=10, D=distances\_matrix,

alpha=1, beta=1, Q=10, tau0=2,

q=0.64)

antResult\_length = get\_len\_t(distances\_matrix, antResult)

nn\_d = np.array(distances\_matrix)

nnResult = nearest\_neighbor(nn\_d)

Окончание листинга 1

nnResult\_length = get\_len\_t(distances\_matrix, nnResult)

print("TSP route: with ants" + str(antResult))

print("TSP route length " + str(antResult\_length))

print("TSP route with NN algorithm" + str(nnResult))

print("TSP route length with NN algorithm " + str(nnResult\_length))

Timer.run(repeat=100)

1. Результаты работы программы

Реализованный муравьиный алгоритм находит решение (одно для одного графа при любом числе повторений), не худшее, чем алгоритм ближайшего соседа (который является эвристическим). Для обоих графов из демонстрации работы была найдена кратчайшая гамильтонова цепь. На рисунках 1 и 2 приведены скриншоты с результатами работы программы для двух графов.

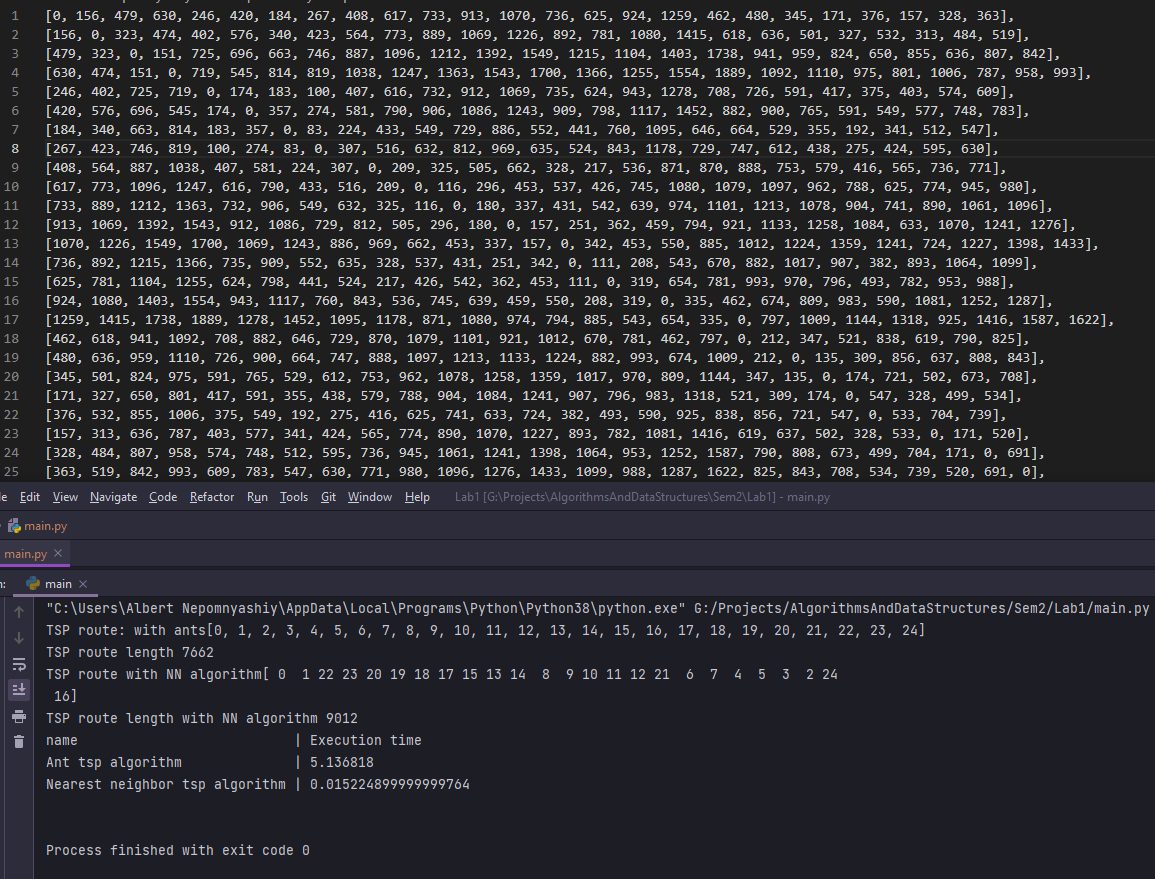


Рисунок 1 – Результат работы программы для графа с 25 вершинами

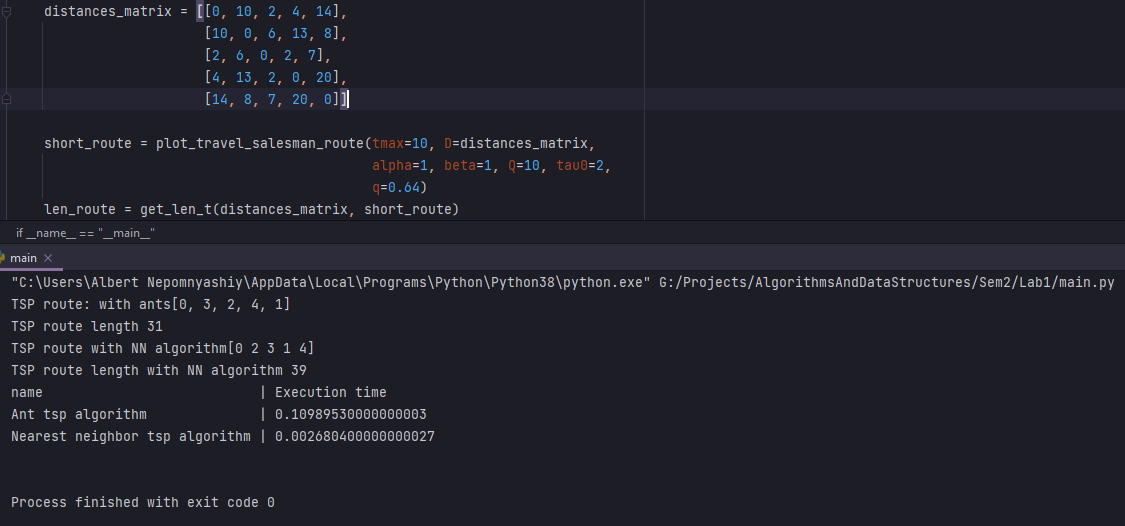


Рисунок 2 – Результат работы программы для графа с 5 вершинами