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. Поиск кратчайшего пути в графе |
| 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 поиска кратчайшего пути в графе с помощью муравьиного алгоритма.

1. Задачи

Требуется разработать программу, которая с помощью муравьиного алгоритма находит кратчайший возможный путь между двумя вершинами графа.

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

Реализован муравьиный алгоритм по поиску кратчайшего пути в графе.

1. Описание программы

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

Поиск кратчайшего пути между вершинами выполняет функция antPath(). Для демонстрации программы была создана матрица весов графа с шестью вершинами.

Для проведения простых тестов производительности алгоритма был использован пакет timethis. Для сравнения был так же взят алгоритм Дейскстры реализации из пакета DijkstraAlgo.

Листинг – Код в файле main.py

import random

from ptimeit import timethis, Timer

import DijkstraAlgo as da

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

len\_t = 0

for l\_i in range(len(T) - 1):

len\_t += D[T[l\_i]][T[l\_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, l\_i, tau, eta, alpha, beta) for l\_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, l\_end, allow\_list, tabu\_list, tau, eta, alpha, beta):

while not tabu\_list or tabu\_list[-1] != l\_end:

allow\_list.remove(cur\_city)

tabu\_list.append(cur\_city)

if cur\_city != l\_end:

build\_route(choose\_next\_city(cur\_city, allow\_list, tau, eta, alpha,

beta),

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

return tabu\_list

Продолжение листинга

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

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

for k in range(m):

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

tabu\_list = []

routes[k] = build\_route(l\_start, l\_end, allow\_list, tabu\_list, tau,

eta, alpha, beta)

return routes

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

for l\_i in range(len(D)):

for j in range(len(D)):

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

for l\_route in routes:

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

for l\_i in range(len(l\_route) - 1):

tau[l\_route[l\_i]][l\_route[l\_i + 1]] += round(delta\_tau, 3)

tau[l\_route[l\_i + 1]][l\_route[l\_i]] += round(delta\_tau, 3)

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

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

return tau

def antPath(tmax, start, end, 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:

tau[i][j] = 0

if start < end:

Продолжение листинга

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

T\_ = [l\_i for l\_i in range(start, end + 1, 1)]

else:

T\_ = [i for i in range(end, start - 1, -1)]

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

for i in range(tmax):

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

for l\_route in routes:

L\_k = get\_len\_t(D, l\_route)

if L\_k < L\_:

T\_ = l\_route

L\_ = L\_k

if i == (tmax - 1):

return T\_

else:

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

@timethis(name="Ant path algorithm")

def benchmarkAntTSP():

antPath(tmax=10, start=source, end=destination,

D=distances\_matrix, alpha=1, beta=1,

Q=10, tau0=2, q=0.64)

return

@timethis(name="Dijkstra's path algorithm")

def benchmarkDijkstra():

dijkstra\_solver = da.DijkstraAlgorithm()

dijkstra\_solver.dijkstraWithPath(distances\_matrix,

source + 1, destination + 1)

return

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

distances\_matrix = [[0, 2, 10, 18, 12, 11],

[2, 0, 23, 3, 12, 14],

[10, 23, 0, 7, 4, 2],

[18, 3, 7, 0, 4, 5],

Окончание листинга

[12, 12, 4, 4, 0, 5],

[11, 14, 2, 5, 5, 0]]

source = 0

destination = 5

x = da.DijkstraAlgorithm()

x.dijkstraWithPath(distances\_matrix, source + 1, destination + 1)

dijkstraRoute = x.path()

dijkstraLength = x.distance()

for i in range(len(dijkstraRoute)):

dijkstraRoute[i] -= 1

if source == destination:

AntRoute = [source, destination]

antLength = 0

else:

AntRoute = antPath(tmax=10, start=source, end=destination,

D=distances\_matrix, alpha=1, beta=1,

Q=10, tau0=2, q=0.64)

antLength = get\_len\_t(distances\_matrix, AntRoute)

print("ANT Route: " + str(AntRoute))

print("ANT Length: " + str(antLength))

print("Dijkstra's Route: " + str(dijkstraRoute))

print("Dijkstra's Length: " + str(dijkstraLength))

Timer.run(repeat=100)

1. Результаты работы программы

Реализованный муравьиный алгоритм находит верное решение (одно для одного графа при любом числе повторений) задачи поиска кратчайшего пути.  
На рисунке 1 приведен скриншот демонстрации работы программы.  
На рисунке 2 приведено изображение графа, к которому был применен алгоритм, и выделен искомый путь.

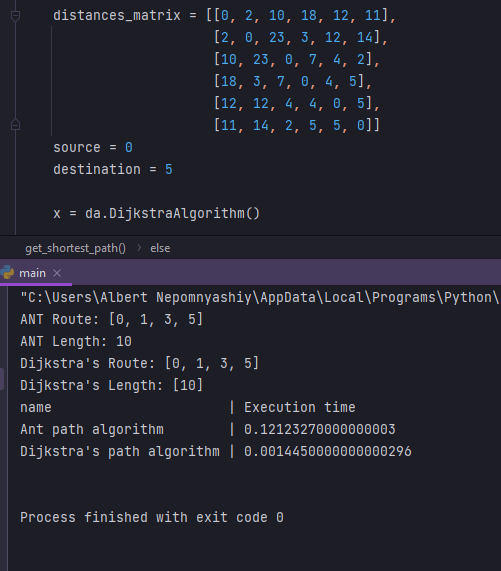


Рисунок 1 – Результат работы программы

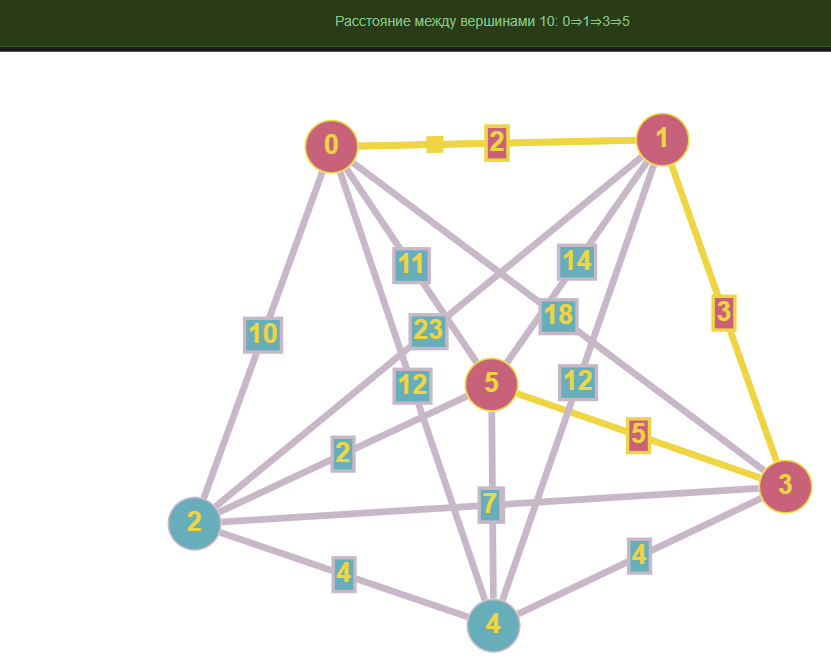


Рисунок 2 – Граф, к которому был применен алгоритм