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.

2 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).

3 Описание реализованного алгоритма

Реализован муравьиный алгоритм по поиску кратчайшего пути в графе.

4 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 Дейкстры 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)
```

5 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.

```
    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],
                        [12, 12, 4, 4, 0, 5],
                        [11, 14, 2, 5, 5, 0]]
    source = 0
    destination = 5

    x = da.DijkstraAlgorithm()
```

get_shortest_path() › else

main

```
"C:\Users\Albert Nepomnyashiy\AppData\Local\Programs\Python\
ANT Route: [0, 1, 3, 5]
ANT Length: 10
Dijkstra's Route: [0, 1, 3, 5]
Dijkstra's Length: [10]
name                    | Execution time
Ant path algorithm      | 0.12123270000000003
Dijkstra's path algorithm | 0.0014450000000000296


Process finished with exit code 0
```

Рисунок 1 – Результат работы программы

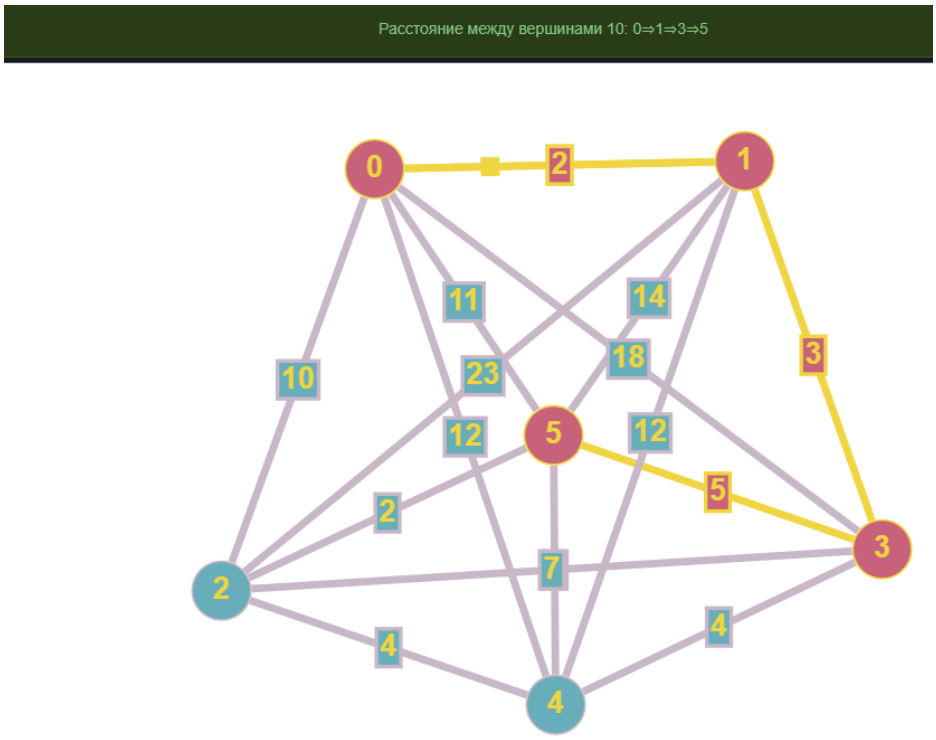

Рисунок 2 – Граф, к которому был применен алгоритм