|  |  |
| --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-01 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Факультет |  | И |  | Информационные и управляющие системы |
|  |  | шифр |  | наименование |
| Кафедра |  | И9 |  | Систем управления и компьютерных технологий |
|  |  | шифр |  | наименование |
| Дисциплина |  | Представление знаний в информационных системах | | |

Отчёт по практической работе

|  |
| --- |
| Логическая модель представления знаний. «Поиск на графах» |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | |  | И508Б |
| Кабиров К. Р. | | | | | |
| Фамилия И.О. | | | | | |
| **РУКОВОДИТЕЛЬ** | | | | | |
|  | |  |  | | |
| Фамилия И.О. Подпись | | | | | |
| Оценка |  | | | |  |
| «\_\_\_\_\_» |  | | | | 2023г. |

**Задание к работе:**

Выполнение лабораторной работы заключается в реализации алгоритмов поиска целевых состояний. Требуется разработать программу, реализующую алгоритм поиска пути (кратчайшего) на предлагаемом графе. Длина пути определяется суммой весов ребер графа.

Исходные данные представлены в виде графа переходов между состояниями среды. Определены вершины, их эвристические оценки (приоритеты), веса ребер графа (стоимость перехода).

Требования к программе:

Программный интерфейс должен обеспечить возможности:

- задать произвольное количество вершин графа (не менее 15).

- назначить каждой вершине эвристическую значимость (численное значение).

- задать произвольное количество ребер графа и назначить им весовые коэффициенты.

- задать начальную вершину графа.

- задать одну целевую вершину графа.

- выбрать способ поиска кратчайшего пути (А\* или «муравьиная колония»).

- вывести протокол поиска и результат поиска (минимальный путь).

- для алгоритма «муравьиной колонии» – задать размер колонии (количество муравьев).

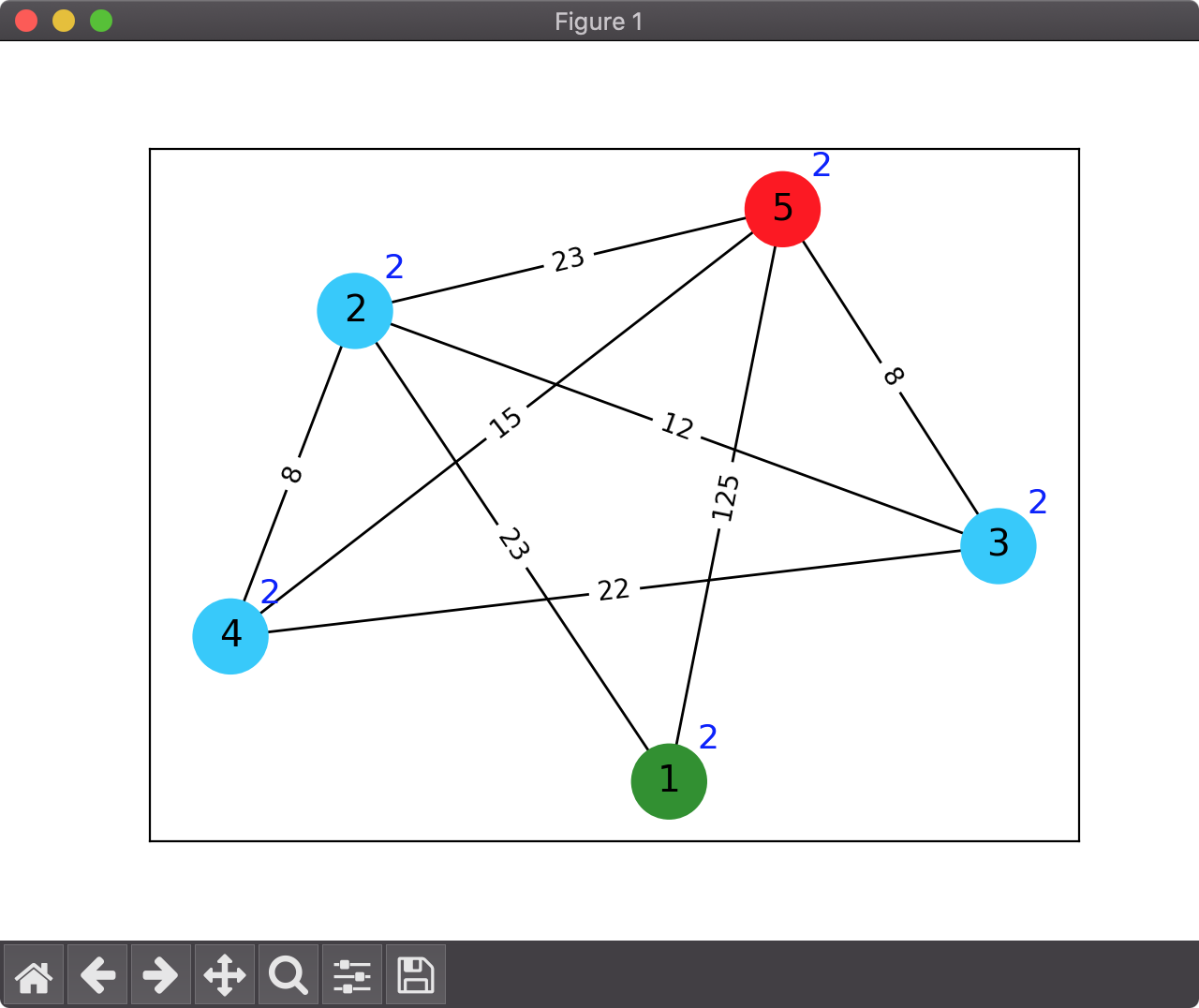
Задание имеет два уровня:

1. Задание на оценку «удовлетворительно» - реализация алгоритма А\*.

2. Дополнительное задание на оценку «отлично» - реализация алгоритма «муравьиной колонии».

1. **Граф, используемый для поиска минимального пути.**

На рисунке 1 представлен граф, используемый для поиска минимального пути.

 Рисунок 1 – Используемый граф

Матрица смежности вышеописанного графа:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 23 | 0 | 0 | 125 |
| 2 | 23 | 0 | 12 | 8 | 23 |
| 3 | 0 | 12 | 0 | 22 | 8 |
| 4 | 0 | 8 | 22 | 0 | 15 |
| 5 | 125 | 23 | 8 | 15 | 0 |

**2 Протокол поиска:**

**2.1 Для алгоритма A\*:**

Вывод протокола:

Шаг 1:

Открытые на этом шаге: {2, 5}

Посещенные: {1}

Шаг 2:

Открытые на этом шаге: {3, 4, 5}

Посещенные: {1, 2}

Шаг 3:

Открытые на этом шаге: {3, 5}

Посещенные: {1, 2, 4}

Шаг 4:

Открытые на этом шаге: {5}

Посещенные: {1, 2, 3, 4}

Кртотчайший путь: 1 -> 2 -> 3 -> 5

Цена пути: 43

**2.2 Для алгоритма муравьиной колонии:**

Состояние графа (фермента на ребрах) после окончания первого, второго и третьего цикла поиска представлены на рисунках 2 и 3:

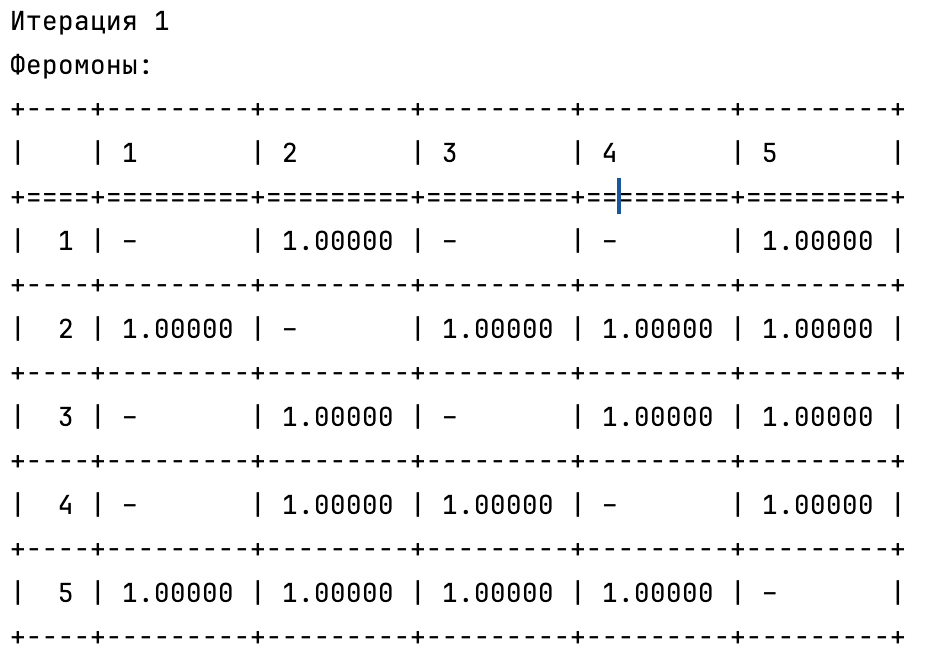


Рисунок 2 – Таблица с феромонами после первой итерации

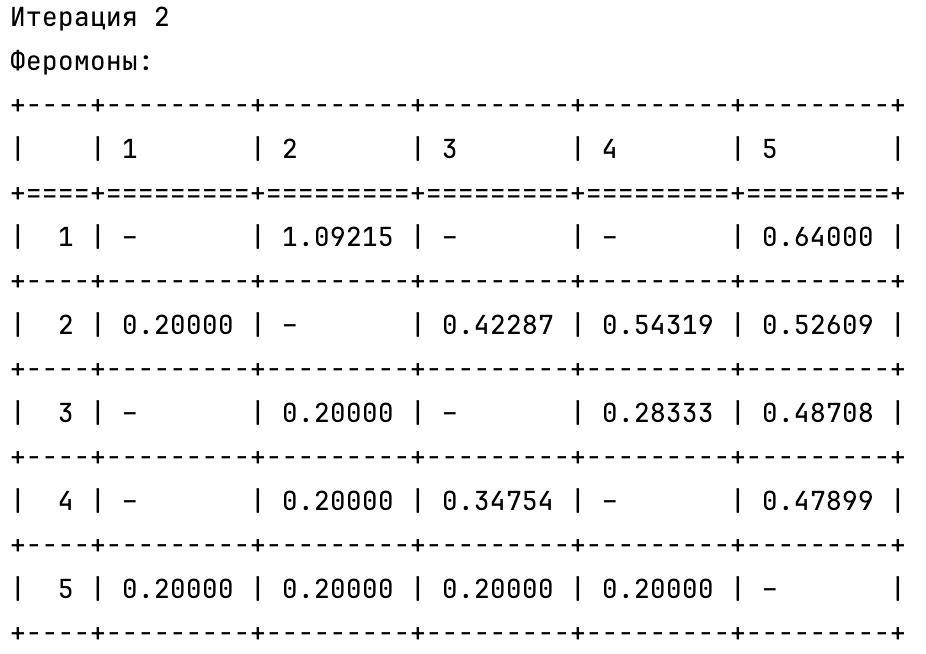


Рисунок 3 – Таблица с феромонами после второй итерации

Пути, пройденные несколькими муравьями по окончании первого цикла поиска:

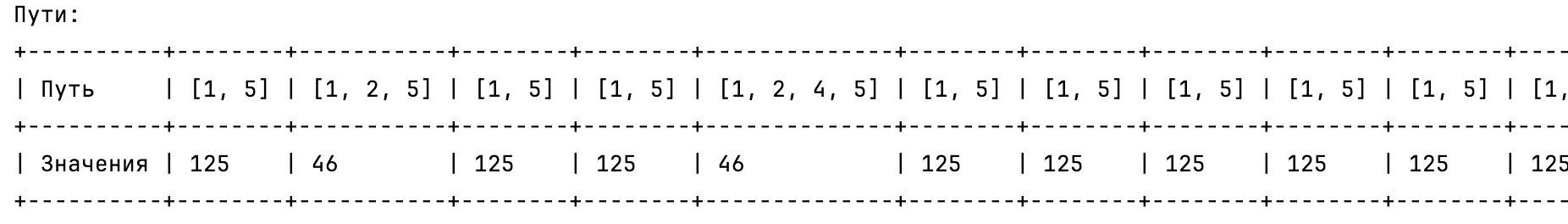


Рисунок 4 – Таблица с путями, которые прошли муравьи

На рисунке 5 представлен протокол поиска на последней итерации и результат работы алгоритма.

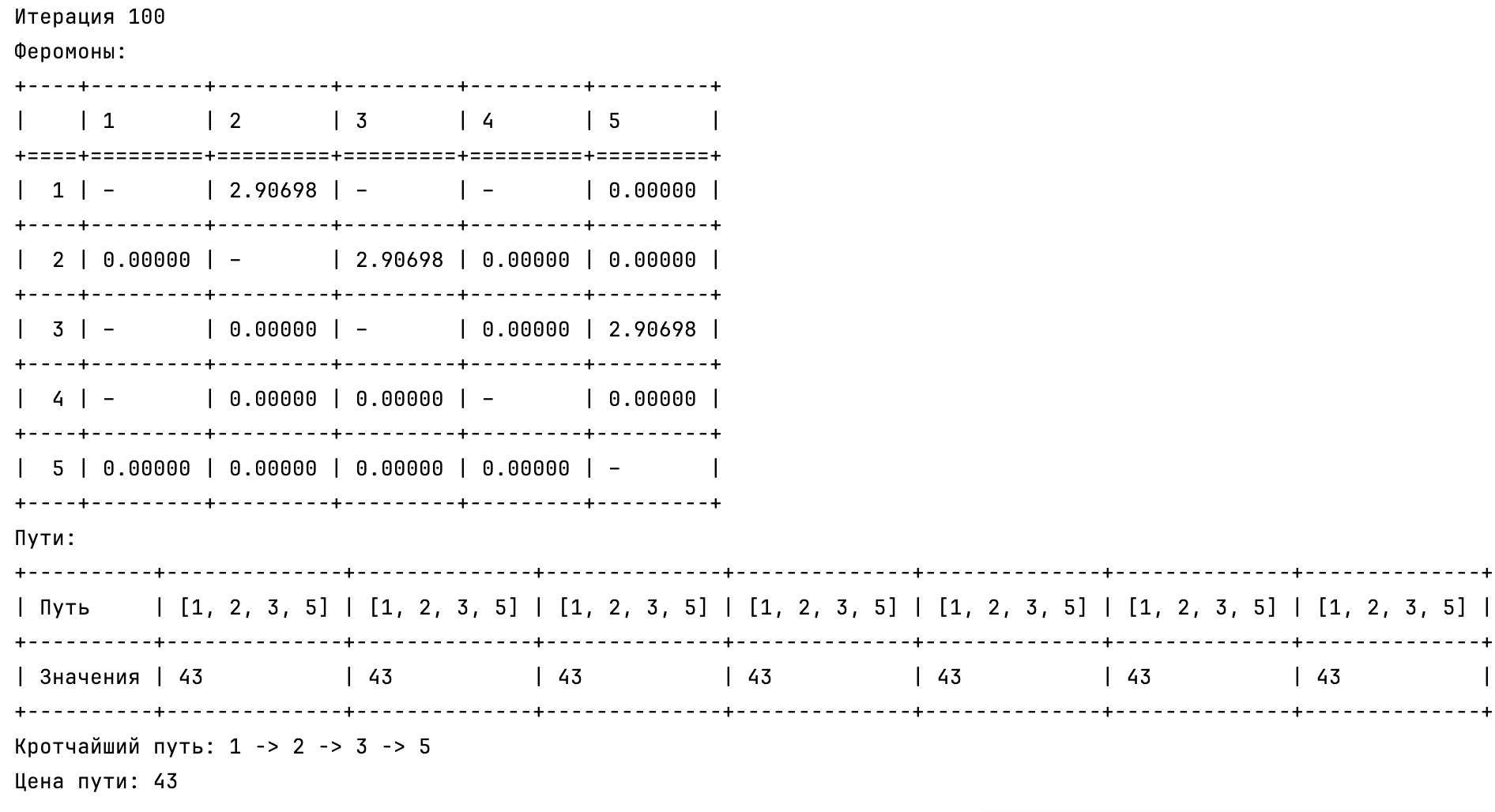


Рисунок 5 – протокол поиска на последней итерации и результат работы алгоритма

**3 Фрагменты программного кода, реализующие алгоритмы поиска (с комментариями).**

**3.1 Для алгоритма A\*:**

def astar(graph, start, goal):

# Initialize the open and closed sets.

open\_set = set([start])

closed\_set = set()

# Initialize the parent and cost variables.

parent = {}

cost = {}

for node in graph:

parent[node] = None

cost[node] = float('inf')

# Set the cost of the start node to 0.

cost[start] = 0

# While the open set is not empty:

while open\_set:

# Find the node with the lowest f(n) value in the open set.

node = min(open\_set, key=lambda n: cost[n] + (graph[n].get('heuristic', 0)))

# Remove the node from the open set.

open\_set.remove(node)

# If the node is the goal node, return the path.

if node == goal:

path = []

while node is not None:

path.append(node)

node = parent[node]

return path[::-1], cost[goal]

# Add the node to the closed set.

closed\_set.add(node)

# For each neighbor of the node:

for neighbor in graph[node]:

# Calculate the tentative cost of the neighbor.

tentative\_cost = cost[node] + graph[node][neighbor]['weight']

# If the tentative cost is less than the current cost of the neighbor:

if tentative\_cost < cost[neighbor]:

# Update the cost of the neighbor.

cost[neighbor] = tentative\_cost

# Set the parent of the neighbor.

parent[neighbor] = node

# Add the neighbor to the open set.

open\_set.add(neighbor)

# Output the contents of the open and unopened vertices at each step of the search.

print("Шаг {}:".format(len(closed\_set)))

print("Открытые на этом шаге: {}".format(open\_set))

print("Посещенные: {}".format(closed\_set))

print()

# The goal node was not found.

return []

**3.2 Для алгоритма муравьиной колонии:**

import random

from tabulate import tabulate

def print\_pher(pheromone):

headers = [''] + list(pheromone.keys())

rows = []

for row\_key, row\_data in pheromone.items():

row = [row\_key] + [f"{row\_data.get(column\_key, '–'):.5f}" if row\_data.get(column\_key) is not None else '–' for

column\_key in headers[1:]]

rows.append(row)

table = tabulate(rows, headers, tablefmt="grid")

print(table)

def print\_paths(paths, path\_lengths):

table\_data = [

["Путь"] + [str(row) for row in paths],

["Значения"] + [str(value) for value in path\_lengths]

]

table = tabulate(table\_data, tablefmt="grid")

print(table)

def ant\_colony\_algorithm(graph, start\_node, end\_node, num\_ants, num\_iterations, evaporation\_rate, alpha, beta, Q):

pheromone = initialize\_pheromone(graph) # Инициализация феромонов на всех ребрах = 1.

best\_path = None

best\_path\_length = float('inf') # Иницилизация кротчайшего пути бесконечностью

search\_log = [] # Журнал поиска, содержащий информацию о каждой итерации.

for i in range(num\_iterations):

paths = []

path\_lengths = []

# Создание пути для каждого муравья

for \_ in range(num\_ants):

path = construct\_path(graph, start\_node, end\_node, pheromone, alpha, beta)

path\_length = calculate\_path\_length(graph, path)

paths.append(path)

path\_lengths.append(path\_length)

# Обновление лучшего пути, если текущий путь лучше

if path\_length < best\_path\_length:

best\_path = path

best\_path\_length = path\_length

print("Итерация", i + 1)

print("Феромоны:")

print\_pher(pheromone)

print("Пути:")

print\_paths(paths, path\_lengths)

# Запись информации о поиске для первых 5 итераций

# if i > :

# Обновление феромонов на ребрах

update\_pheromone(pheromone, paths, path\_lengths, evaporation\_rate, Q)

return best\_path, best\_path\_length

def initialize\_pheromone(graph):

pheromone = {}

for node in graph:

pheromone[node] = {}

for neighbor in graph[node]:

pheromone[node][neighbor] = 1.0

return pheromone

def construct\_path(graph, start\_node, end\_node, pheromone, alpha, beta):

# Построение пути муравья от начальной до конечной вершины

path = [start\_node]

current\_node = start\_node

while current\_node != end\_node:

probabilities = calculate\_probabilities(graph, current\_node, path, pheromone, alpha, beta)

next\_node = select\_next\_node(probabilities)

path.append(next\_node)

current\_node = next\_node

return path

def calculate\_probabilities(graph, current\_node, path, pheromone, alpha, beta):

# Вычисление вероятностей перехода муравья в соседние вершины

probabilities = []

total = 0.0

for neighbor in graph[current\_node]:

if neighbor not in path:

pheromone\_level = pheromone[current\_node][neighbor]

heuristic = graph[current\_node][neighbor]['heuristic']

probability = pheromone\_level \*\* alpha \* heuristic \*\* beta

probabilities.append((neighbor, probability))

total += probability

# Нормализация вероятностей

probabilities = [(neighbor, probability / total) for neighbor, probability in probabilities]

return probabilities

def select\_next\_node(probabilities):

# Выбор следующей вершины на основе вероятностей

random\_value = random.uniform(0, 1)

cumulative\_probability = 0.0

for neighbor, probability in probabilities:

cumulative\_probability += probability

if random\_value <= cumulative\_probability:

return neighbor

def calculate\_path\_length(graph, path):

# Вычисление длины пути

length = 0

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

node = path[i]

next\_node = path[i + 1]

length += graph[node][next\_node]['weight']

return length

def update\_pheromone(pheromone, paths, path\_lengths, evaporation\_rate, Q):

# Обновление феромонов на ребрах

for node in pheromone:

for neighbor in pheromone[node]:

pheromone[node][neighbor] \*= (1 - evaporation\_rate)

for i in range(len(paths)):

path = paths[i]

path\_length = path\_lengths[i]

for j in range(len(path) - 1):

node = path[j]

next\_node = path[j + 1]

pheromone[node][next\_node] += Q / path\_length