Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Искусственный интеллект в профессиональной сфере»
Вариант №1

Выполнила:
Беседина Инга Олеговна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Проверил:
Воронкин Р. А., канд. технических
наук, доцент, доцент департамента
цифровых, робототехнических систем
и электроники института
перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Исследование методов поиска в пространстве состояний

Цель: Приобретение навыков по работе с методами поиска в пространстве состояний с помощью языка программирования Python версии 3.x

Ход работы:

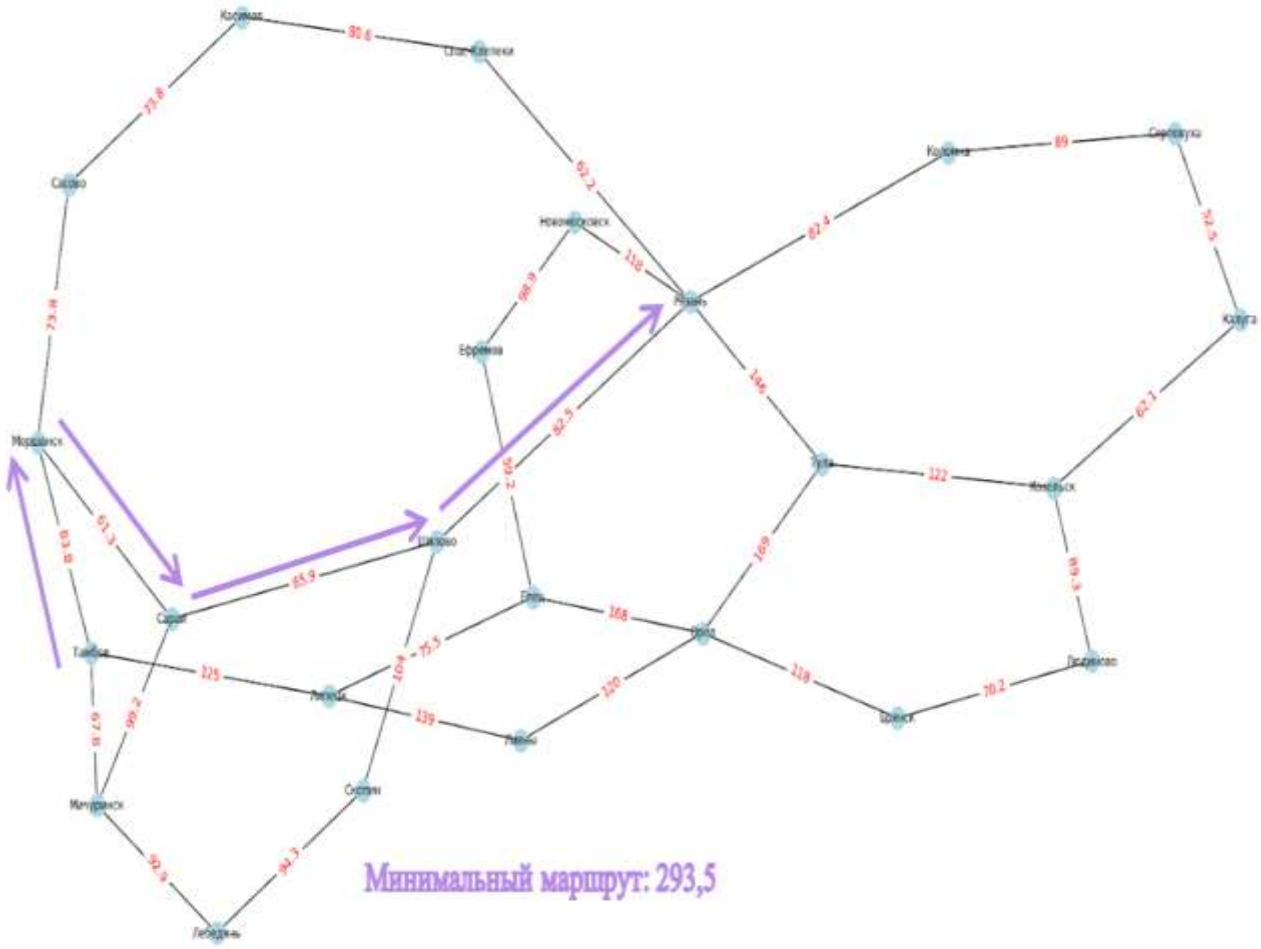

Рисунок 1. Построенный граф

Решение задачи коммивояжёра методом полного перебора:

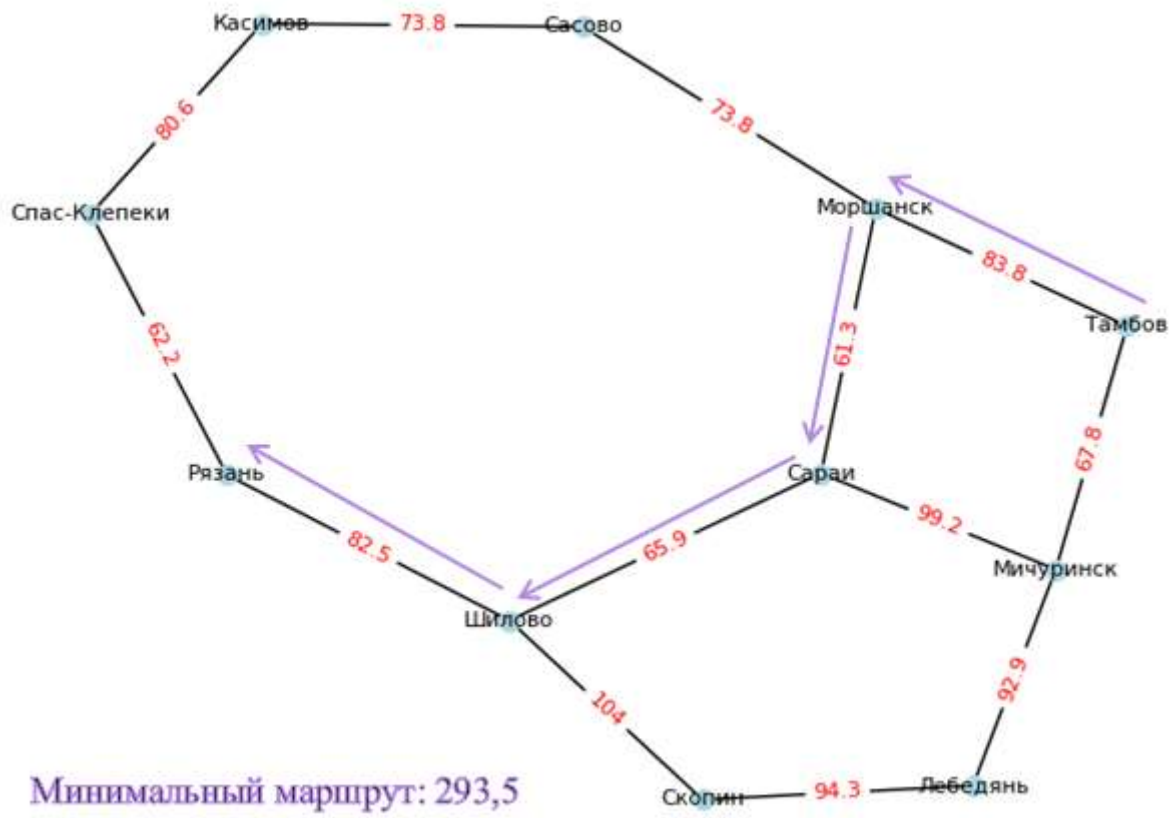

Рисунок 2. Граф для решения задачи методом полного перебора

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from itertools import permutations


graph = {
'Тамбов': {'Мичуринск': 67.8, 'Моршанск': 83.8},
'Мичуринск': {'Тамбов': 67.8, 'Сараи': 99.2, 'Лебедянь': 92.9},
'Лебедянь': {'Мичуринск': 92.9, 'Скопин': 94.3},
'Скопин': {'Лебедянь': 94.3, 'Шилово': 104},
'Шилово': {'Сараи': 65.9, 'Скопин': 104, 'Рязань': 82.5},
'Сараи': {'Моршанск': 61.3, 'Мичуринск': 99.2, 'Шилово': 65.9},
'Моршанск': {'Тамбов': 83.8, 'Сараи': 61.3, 'Сасово': 73.8},
'Сасово': {'Моршанск': 73.8, 'Касимов': 73.8},
'Касимов': {'Сасово': 73.8, 'Спас-Клепеки': 80.6},
'Спас-Клепеки': {'Касимов': 80.6, 'Рязань': 62.2},
'Рязань': {'Спас-Клепеки': 62.2, 'Шилово':  82.5}
}
```

```python
def calc_distance(route):
    """Функция для вычисления расстояния маршрута."""
    distance = 0
    for i in range(len(route) - 1):
        distance += graph[route[i]].get(route[i + 1], float('inf'))
    return distance

def find_min_route(start, end):
    """Функция для нахождения минимального маршрута с использованием
полного перебора."""
    nodes = list(graph.keys())
    nodes.remove(start)
    nodes.remove(end)

    min_distance = float('inf')
    best_route = None

    # Перебор всех возможных маршрутов
    for r in range(len(nodes) + 1):
        for perm in permutations(nodes, r):
            route = [start] + list(perm) + [end]
            distance = calc_distance(route)

            if distance < min_distance:
                min_distance = distance
                best_route = route

    return best_route, min_distance


def main():
    start_point = 'Тамбов'
    end_point = 'Рязань'

    best_route, min_distance = find_min_route(start_point, end_point)

    print(f"Лучший маршрут от {start_point} до {end_point}: {' ->
'.join(best_route)}")
    print(f"Минимальное расстояние: {min_distance}")


if __name__ == "__main__":
    main()
```

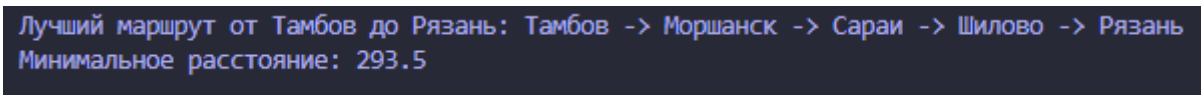

Рисунок 3. Результат работы программы

Контрольные вопросы:

1. Метод "слепого поиска" исследует пространство возможных решений методом проб и ошибок, не обладая информацией о том, насколько близко каждое принятое решение к финальной цели

2. Эвристический поиск, в отличие от слепого, использует дополнительные знания или "эвристики" для направления процесса поиска

3. Эвристика - дополнительные знания, которые используются в процессе поиска

4. Один из наиболее известных примеров эвристического поиска - алгоритм A*, который находит наиболее оптимальный путь к цели, основываясь на заранее заданных критериях и предположениях

5. Разработка ИИ для игры в шахматы требует глубокого понимания не только правил, но и бесчисленных стратегий и тактик

6. Из-за огромного количества возможных ходов в шахматах, полное исследование всех вариантов становится практически невозможным. Даже если технически возможно построить структуру данных для представления всех возможных ходов в шахматах, встает вопрос о ресурсах - времени и памяти

7. Основная задача искусственного интеллекта при выборе ходов в шахмата заключается в быстрой обрабатке информации и принятии решения в условиях когда время может быть ограничено

8. Некоторые алгоритмы могут быстро находить решения, но они могут быть далеки от оптимальных. Другие, напротив, способны находить наилучшие решения, но требуют значительных временных и ресурсных затрат

9. Основные элементы задачи поиска маршрута по карте: исходная точка и конечная точка, графическая модель карты, вес ребер, алгоритм поиска, критерий оптимальности

10. Оптимальность решения в данном случае предполагает нахождение маршрута с минимальной стоимостью

11. Исходное состояние дерева поиска в задаче маршрутизации по карте Румынии - исходная точка (город Арад)

12. В контексте алгоритма поиска по дереву листовыми узлами (или просто листьями) называются узлы, которые не имеют дочерних узлов. Это означает, что они находятся на самом нижнем уровне дерева и не ведут к другим узлам

13. На этапе расширения узла в дереве поиска происходит генерация дочерних узлов

14. Сибиу, Тимишоара или Зеринд

15. Целевое состояние в алгоритме поиска по дереву определяется в зависимости от конкретной задачи, которую решает алгоритм. Обычно это состояние соответствует условию, которое необходимо выполнить для завершения поиска

16. Основные шаги, которые выполняет алгоритм поиска по дереву: инициализация (создание корневого узла дерева), проверка целевого состояния, генерация дочерних узлов, добавление дочерних узлов в структуру данных, выбор следующего узла для обработки, проверка целевого состояния

17. Состояние - это представление конфигурации в нашем пространстве поиска, например, города на карте Румынии, где мы находимся, или конфигурация плиток в головоломке, а узел - это структура данных о состоянии, содержащая само состояние, а также другие данные: указатель на родительский узел, действие, стоимость пути и глубину узла в дереве

18. Функция преемника — это функция, используемая в алгоритмах поиска, которая возвращает все возможные состояния (или узлы), которые могут быть достигнуты из текущего состояния. Она генерирует "потомков" (или "преемников") для заданного узла, позволяя алгоритму исследовать пространство состояний

19. b (максимальный коэффициент разветвления) - показывает, сколько дочерних узлов может иметь один узел. d (глубина наименее дорогого решения) - определяет, насколько далеко нужно спуститься по дереву для нахождения оптимального решения. m (максимальная глубина дерева) - показывает, насколько глубоко можно в принципе спуститься по дереву. В некоторых случаях это значение может быть бесконечным

20. Полнота означает, что алгоритм находит решение, если оно существует. Отсутствие решения возможно только в случае, когда задача невыполнима. Временная сложность измеряется количеством сгенерированных узлов, а не временем в секундах или циклах ЦПУ. Она пропорциональна общему количеству узлов. Пространственная сложность относится к максимальному количеству узлов, которые нужно хранить в памяти в любой момент времени. В некоторых алгоритмах она совпадает с временной сложностью.

21. Класс Problem служит шаблоном для создания конкретных задач в различных предметных областях. Каждая конкретная задача будет наследовать этот класс и переопределять его методы

22. Методы action, result, is_goal, action_cost, h необходимо переопределить при наследовании класса Problem

23. Метод is_goal проверяет, достигнуто ли целевое состояние

24. Методы action_cost и h предоставляют стандартные реализации для стоимости действия и эвристической функции соответственно.

25. Класс Node представляет узел в дереве поиска

26. Конструктор класса Node принимает такие параметры как: текущее состояние (state), ссылку на родительский узел (parent), действие, которое привело к этому узлу (action), и стоимость пути (path_cost)

27. Узел failure создаётся для обозначения неудачи в поиске

28. Функция expand расширяет узел, генерируя дочерние узлы

29. Функция path_actions возвращает последовательность действий, которые привели к данному узлу

30. Функция path_states возвращает последовательность состояний, ведущих к данному узлу

31. FIFO (First In, First Out) очередь обычно реализуется с помощью списков (например, list в Python) или двусвязных списков. В Python также можно использовать модуль collections.deque, который обеспечивает эффективные операции добавления и удаления элементов с обеих сторон

32. В FIFOQueue элементы обрабатываются в порядке их добавления. Первый добавленный элемент будет первым, который будет извлечен. В LIFOQueue последний добавленный элемент будет первым, который будет извлечен

33. В классе PriorityQueue метод add добавляет элемент в очередь с приоритетом. При этом элемент помещается в структуру данных так, чтобы порядок обработки соответствовал его приоритету

34. Очереди с приоритетом применяются в различных ситуациях:

Управление задачами: В операционных системах для планирования процессов, где задачи с более высоким приоритетом должны выполняться первыми.

Алгоритмы поиска: Например, в алгоритме Дейкстры для нахождения кратчайшего пути.

Системы обработки событий: Где события с более высоким приоритетом должны быть обработаны раньше.

Игровые движки: Для управления событиями и действиями на основе их важности

35. Функция heappop используется для извлечения элемента с наивысшим приоритетом из кучи (heap). Она поддерживает свойства кучи: после извлечения элемента heappop автоматически перестраивает кучу, чтобы сохранить ее свойства (наименьший или наибольший элемент всегда находится на вершине). Операция heappop выполняется за время O(log n), что делает ее эффективной для работы с большими объемами данных.

Вывод: В ходе выполнения лабораторной работы были приобретены навыки по работе с методами поиска в пространстве состояний с помощью языка программирования Python версии 3.x