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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

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

Выполнила:
Михеева Елена Александровна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Воронкин Роман Александрович

_____
(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

Порядок выполнения работы:

Репозиторий: https://github.com/helendddd/Ai_1.git **.**

Был построен граф из 20 населенных пунктов Италии. Узлы данного графа представляют населённые пункты, а рёбра — дороги, соединяющие их. Вес каждого ребра соответствует расстоянию между этими пунктами.

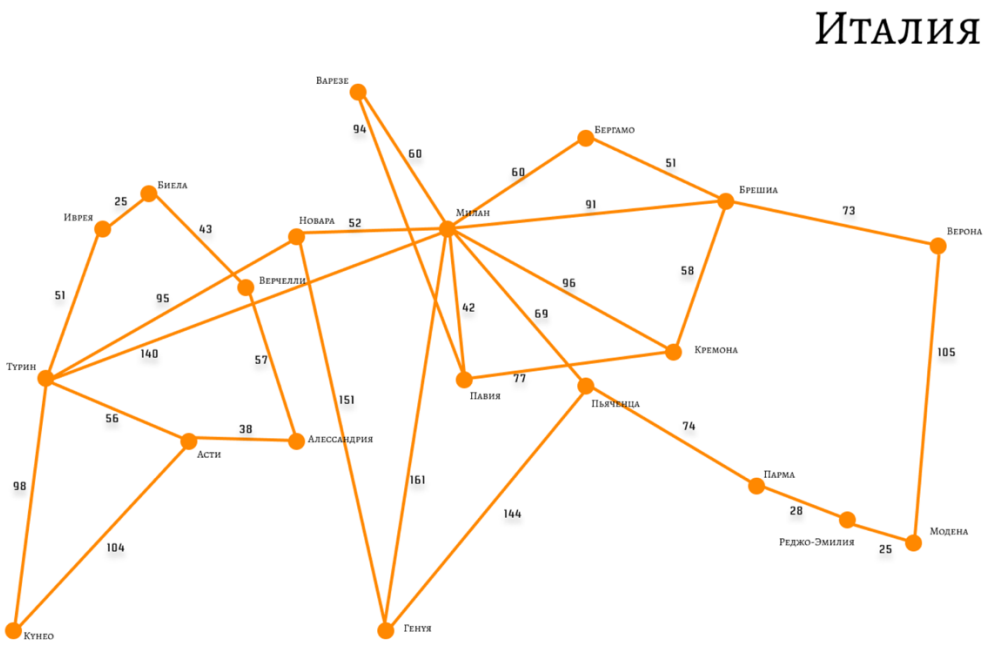
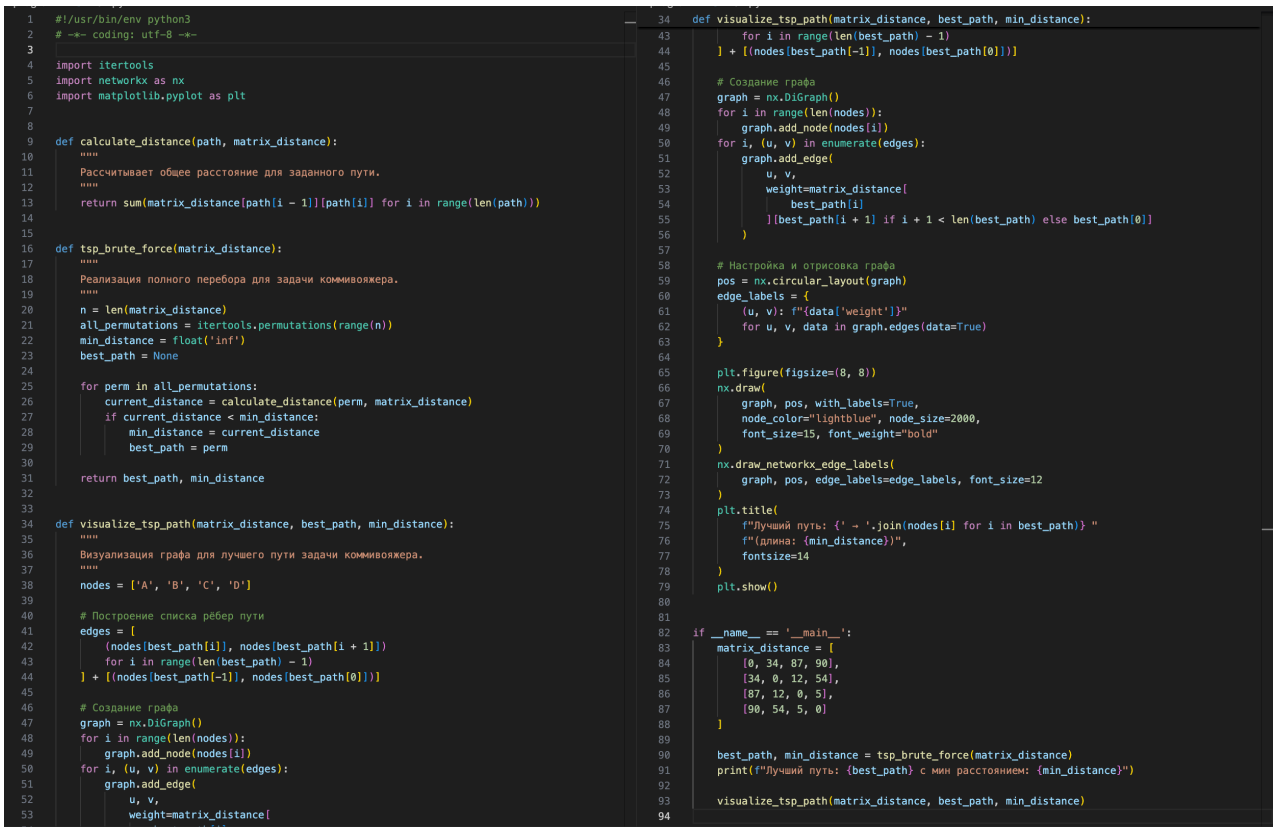


Рисунок 1. Граф из населенных пунктов

Методом полного перебора была решена задача коммивояжера для начального населенного пункта на построенном графе.



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

import itertools
import networkx as nx
import matplotlib.pyplot as plt


def calculate_distance(path, matrix_distance):
    """
    Рассчитывает общее расстояние для заданного пути.
    """
    return sum(matrix_distance[path[i - 1]][path[i]] for i in range(len(path)))


def tsp_brute_force(matrix_distance):
    """
    Реализация полного перебора для задачи коммивояжера.
    """
    n = len(matrix_distance)
    all_permutations = itertools.permutations(range(n))
    min_distance = float('inf')
    best_path = None

    for perm in all_permutations:
        current_distance = calculate_distance(perm, matrix_distance)
        if current_distance < min_distance:
            min_distance = current_distance
            best_path = perm

    return best_path, min_distance


def visualize_tsp_path(matrix_distance, best_path, min_distance):
    """
    Визуализация графа для лучшего пути задачи коммивояжера.
    """
    nodes = ['A', 'B', 'C', 'D']

    # Построение списка рёбер пути
    edges = [
        (nodes[best_path[i]], nodes[best_path[i + 1]])
        for i in range(len(best_path) - 1)
    ] + [(nodes[best_path[-1]], nodes[best_path[0]])]

    # Создание графа
    graph = nx.DiGraph()
    for i in range(len(nodes)):
        graph.add_node(nodes[i])
    for i, (u, v) in enumerate(edges):
        graph.add_edge(
            u, v,
            weight=matrix_distance[
                best_path[i]
            ][best_path[i + 1] if i + 1 < len(best_path) else best_path[0]]
        )

    # Настройка и отрисовка графа
    pos = nx.circular_layout(graph)
    edge_labels = {
        (u, v): f"{data['weight']}"
        for u, v, data in graph.edges(data=True)
    }

    plt.figure(figsize=(8, 8))
    nx.draw(
        graph, pos, with_labels=True,
        node_color="lightblue", node_size=2000,
        font_size=15, font_weight="bold"
    )
    nx.draw_networkx_edge_labels(
        graph, pos, edge_labels=edge_labels, font_size=12
    )
    plt.title(
        f"Лучший путь: {' → '.join(nodes[i] for i in best_path)} "
        f"(длина: {min_distance})",
        fontsize=14
    )
    plt.show()


if __name__ == '__main__':
    matrix_distance = [
        [0, 34, 87, 90],
        [34, 0, 12, 54],
        [87, 12, 0, 5],
        [90, 54, 5, 0]
    ]

    best_path, min_distance = tsp_brute_force(matrix_distance)
    print(f"Лучший путь: {best_path} с мин расстоянием: {min_distance}")

    visualize_tsp_path(matrix_distance, best_path, min_distance)
```

Рисунок 2. Код программы

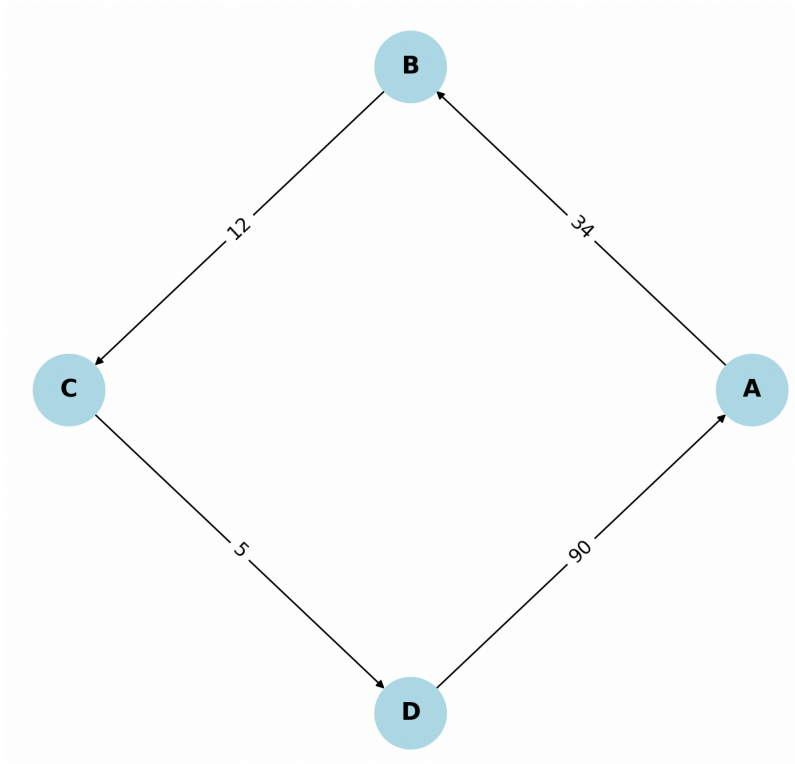


Рисунок 3. Результат работы программы

Ответы на контрольные вопросы:

1. Что представляет собой метод "слепого поиска" в искусственном интеллекте?

Самым базовым методом является "слепой поиск". Этот метод можно представить как попытку найти выход из затемненного лабиринта, ощупывая каждую стену, каждый угол, без заранее известного плана или карты. Он исследует пространство возможных решений методом проб и ошибок, не обладая информацией о том, насколько близко каждое принятое решение к финальной цели.

2. Как отличается эвристический поиск от слепого поиска?

Эвристический поиск, в отличие от слепого, использует дополнительные знания или "эвристики" для направления процесса поиска, подобно тому, как путешественник использует компас в неизведанных землях.

3. Какую роль играет эвристика в процессе поиска?

Основная задача эвристики сводится к построению моделей осуществления процесса поиска нового для данного субъекта (или общества в целом) решения задачи.

4. Приведите пример применения эвристического поиска в реальной задаче.

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

5. Почему полное исследование всех возможных ходов в шахматах затруднительно для ИИ?

Центральная задача в создании шахматного ИИ — это выбор оптимальных ходов. В идеальном мире, программа могла бы анализировать каждый возможный ход и его последствия, строить огромное дерево всех возможных комбинаций и выбирать наиболее перспективные варианты. Учитывая огромное количество возможных ходов в шахматах, полное исследование всех вариантов становится практически невозможным даже для

современных суперкомпьютеров. Это требует от разработчиков ИИ использования сложных стратегий для эффективного просеивания и анализа ходов, отсекая менее перспективные и сосредотачиваясь на более целесообразных.

6. Какие факторы ограничивают создание идеального шахматного ИИ?

Даже если технически возможно построить структуру данных для представления всех возможных ходов в шахматах, встает вопрос о ресурсах - времени и памяти. Для многих задач эти ограничения могут быть преодолены, но в контексте шахмат они становятся критическими факторами. В чем заключается основная задача искусственного интеллекта при выборе ходов в шахматах?

7. Как алгоритмы ИИ балансируют между скоростью вычислений и нахождением оптимальных решений?

Необходимость быстро обрабатывать информацию и принимать решения в условиях, когда время может быть ограниченно делает задачу создания эффективного шахматного ИИ особенно сложной.

8. Каковы основные элементы задачи поиска маршрута по карте?

Основные элементы задачи поиска маршрута включают: начальную точку, целевую точку, граф (состоящий из узлов и рёбер), метод оценки расстояний и ограничения.

9. Как можно оценить оптимальность решения задачи маршрутизации на карте Румынии?

Оптимальность решения можно оценить по критериям, таким как минимальная длина маршрута, минимальное время в пути или минимальная стоимость.

10. Что представляет собой исходное состояние дерева поиска в задаче маршрутизации по карте Румынии?

Исходное состояние дерева поиска – это корневой узел, представляющий начальную точку маршрута.

11. Какие узлы называются листовыми в контексте алгоритма поиска по дереву?

Листовые узлы – это узлы, которые не имеют дочерних узлов; они являются конечными состояниями в дереве поиска.

12. Что происходит на этапе расширения узла в дереве поиска?

Применяется функция преемника, узел прекращает быть листовым узлом, и в дереве появляются новые листовые узлы.

13. Какие города можно посетить, совершив одно действие из Арада в примере задачи поиска по карте?

Сибиу, Тимишоара, Зеринд.

14. Как определяется целевое состояние в алгоритме поиска по дереву?

Достижение целевого узла это и есть целевое состояние.

15. Какие основные шаги выполняет алгоритм поиска по дереву?

По алгоритму поиска по дереву первым шагом будет выбор листового узла для дальнейшего расширения.

Сначала мы проверяем, достигли ли мы цели. В данном случае — нет,

поскольку исходное состояние не совпадает с целевым. Следовательно, мы расширяем исходный узел. Расширение подразумевает применение функциипреемника, которая определяет все возможные действия, применимые к текущему состоянию. Эта функция генерирует дочерние узлы,

представляющие все города, в которые можно попасть, совершив одно

действие из текущего.

16. Чем различаются состояния и узлы в дереве поиска?

Состояние (State) - это представление конфигурации в нашем пространстве поиска, например, города на карте Румынии, где мы находимся, или конфигурация плиток в головоломке.

Узел (Node) - это структура данных о состоянии, содержащая само состояние, а также другие данные: указатель на родительский узел, действие, стоимость пути и глубину узла в дереве.

17. Что такое функция преемника и как она используется в алгоритме поиска?

Расширение подразумевает применение функции преемника, которая определяет все возможные действия, применимые к текущему состоянию. Эта функция генерирует дочерние узлы, представляющие все города, в которые можно попасть, совершив одно действие из текущего.

18. Какое влияние на поиск оказывают такие параметры, как b (разветвление), d (глубина решения) и m (максимальная глубина)?

b, d и m используются для описания временной и пространственной сложности в алгебраической форме, позволяя точно оценить как изменяется количество сгенерированных узлов в зависимости от параметров задачи.

19. Как алгоритмы поиска по дереву оцениваются по критериям полноты, временной и пространственной сложности, а также оптимальности?

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.

Пространственная сложность зависит от того, какие узлы сохраняются в памяти. В некоторых алгоритмах необходимо сохранять все сгенерированные узлы, тогда как в других узлы могут быть отброшены после проверки, не требуя длительного сохранения в памяти. Это различие позволяет временной и пространственной сложности отличаться друг от друга.

Оптимальность - это способность алгоритма всегда находить наилучшее решение с минимальной стоимостью, если таковое существует. Она не связана напрямую с временной или пространственной сложностью, которые измеряют количество узлов. Оптимальность не гарантирует быстродействие или экономию памяти, а лишь обеспечивает нахождение решения с наименьшей стоимостью.

1. b (максимальный коэффициент разветвления) - показывает, сколько дочерних узлов может иметь один узел.

2. d (глубина наименее дорогого решения) - определяет, насколько далеко нужно спуститься по дереву для нахождения оптимального решения.

3. m (максимальная глубина дерева) - показывает, насколько глубоко можно в принципе спуститьс

20. Какую роль выполняет класс Problem в приведенном коде?

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

21. Какие методы необходимо переопределить при наследовании класса Problem?

Методы actions, result, is_goal, action_cost и h.

22. Что делает метод is_goal в классе Problem?

Метод is_goal проверяет, достигнуто ли целевое состояние.

23. Для чего используется метод action_cost в классе Problem?

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

24. Какую задачу выполняет класс Node в алгоритмах поиска?

Node представляет узел в дереве поиска.

25. Какие параметры принимает конструктор класса Node?

Принимает текущее состояние (state), ссылку на родительский узел (parent), действие, которое привело к этому узлу (action), и стоимость пути (path_cost).

26. Что представляет собой специальный узел failure?

Обозначение неудачи в поиске.

27. Для чего используется функция expand в коде?

Расширяет узел, генерируя дочерние узлы.

28. Какая последовательность действий генерируется с помощью функции path_actions?

Та, с помощью которой можно добраться до узла node.

29. Чем отличается функция path_states от функции path_actions?

path_actions дает последовательность действий, чтобы добраться до узла node, а path_states - последовательность состояний

30. Какой тип данных используется для реализации FIFOQueue?

Она реализуется с помощью deque из модуля collections . deque – это обобщенная версия стека и очереди, которая поддерживает добавление и удаление элементов с обоих концов.

31. Чем отличается очередь FIFOQueue от LIFOQueue?

FIFOQueue - это очередь, где первый добавленный элемент будет первым извлеченным.

LIFOQueue - это стек, где последний добавленный элемент будет первым извлеченным.

32. Как работает метод add в классе PriorityQueue?

Каждый элемент добавляется в кучу с его приоритетом, определенным функцией key .

33. В каких ситуациях применяются очереди с приоритетом?

Это полезно, когда нужно обработать наиболее важные элементы в первую очередь.

34. Как функция heappop помогает в реализации очереди с приоритетом?

Когда необходима гарантия извлечения элемента с наименьшим приоритетом.

Вывод: выполняя данную работу были получены навыки по работе с методами поиска в пространстве состояний с помощью языка программирования Python версии 3.x.