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»
Вариант___

Выполнил:
Епифанов Алексей Александрович
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: элементы объектно-ориентированного программирования в языке Python.

Цель: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

Репозиторий: https://github.com/alexeiepif/ii_1.git

2. Выполнил задание: воспользуйтесь сервисом Google Maps или аналогичными картографическими приложениями, чтобы выбрать на карте более 20 населённых пунктов, связанных между собой дорогами. Постройте граф, где узлы будут представлять населённые пункты, а рёбра — дороги, соединяющие их. Вес каждого ребра соответствует расстоянию между этими пунктами. Выберите начальный и конечный пункты на графе. Определите минимальный маршрут между ними, который должен проходить через три промежуточных населённых пункта. Покажите данный путь на построенном графе.

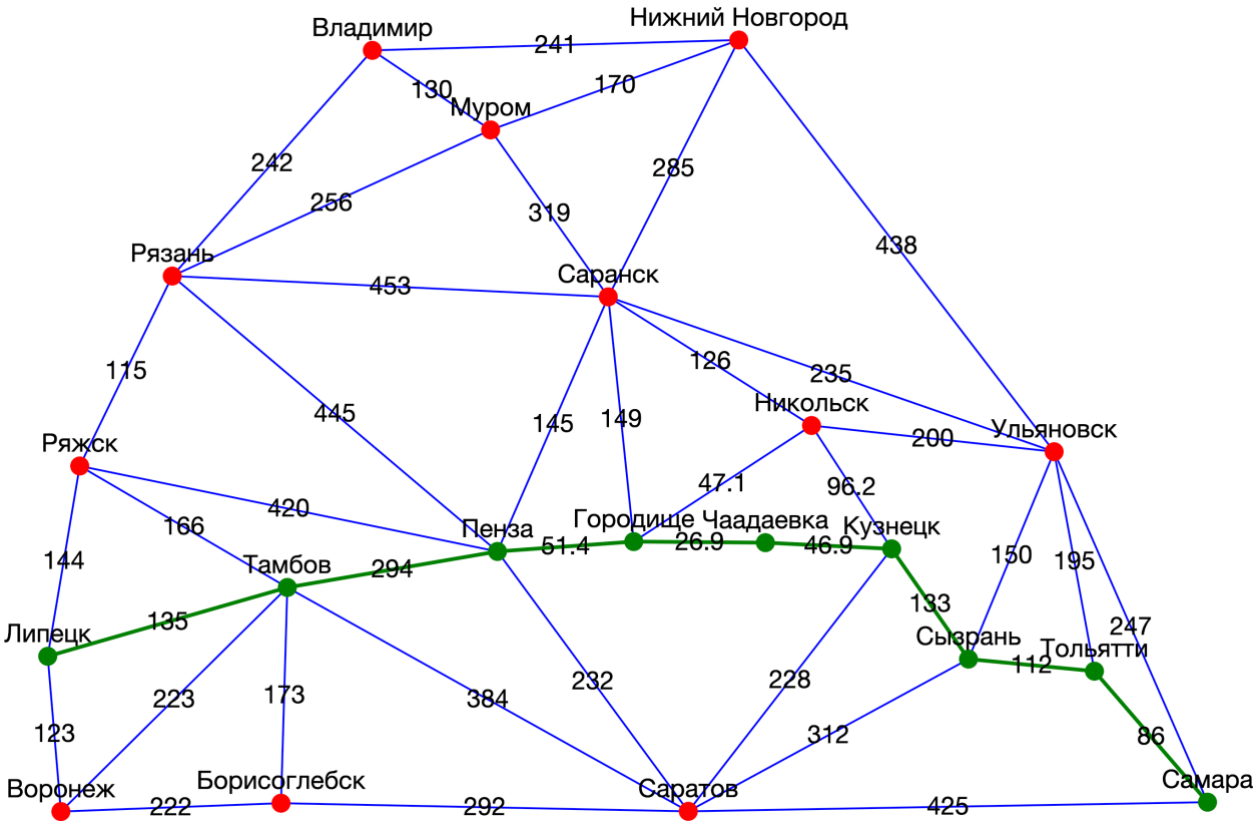

Рисунок 1. Полный граф с кратчайшим путем

Кратчайшее расстояние от Липецка до Самары: 885.2км

3. Выполнил задание: Методом полного перебора решите задачу коммивояжёра.

Рисунок 2. Усеченный граф, использующийся в решении задачи

```
сем/ИИвПС/code/ii_1/program/py.py
[[  0.   51.4   nan   nan   nan 145.    nan 232.    nan   nan]
 [ 51.4   0.   26.9   nan  47.1 149.    nan   nan   nan   nan]
 [  nan  26.9   0.   46.9   nan   nan   nan   nan   nan   nan]
 [  nan   nan  46.9   0.   96.2   nan 133.  228.    nan   nan]
 [  nan  47.1   nan  96.2   0.  126.    nan   nan 200.    nan]
 [145.  149.    nan   nan 126.    0.    nan   nan 235.    nan]
 [  nan   nan   nan 133.    nan   nan   0.  312.  150.  112. ]
 [232.    nan   nan 228.    nan   nan 312.    0.    nan   nan]
 [  nan   nan   nan   nan 200.  235.  150.    nan   0.  195. ]
 [  nan   nan   nan   nan   nan   nan 112.    nan 195.    0. ]]
Самый короткий путь: ['Пенза', 'Городище', 'Чаадаевка', 'Кузнецк', 'Никольск', 'Саранск', 'Ульяновск', 'Тольятти', 'Сызрань', 'Саратов']
Длина самого короткого пути: 1433.4
```

Рисунок 3. Результат работы программы

Ответы на контрольные вопросы:

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

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. Такой подход, хотя и элементарен, является основой для разработки более сложных алгоритмов, включая эвристический поиск.

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

3. Какую роль играет эвристика в процессе поиска?

Эвристика - дополнительные знания для направления процесса поиска.

4. Приведите пример применения эвристического поиска в реальной задаче.

Задача заключается в нахождении кратчайшего пути для робота, перемещающегося по лабиринту, представленному в виде двумерной сетки. Лабиринт состоит из свободных для перемещения клеток и препятствий, которые необходимо обходить. Начальная и конечная точки задаются как координаты на сетке, и требуется определить оптимальный маршрут между ними.

Эвристический поиск, такой как алгоритм A*, может быть использован для решения этой задачи. Он использует комбинацию эвристической оценки и фактического расстояния для нахождения оптимального пути. Эвристика в данном случае может быть оценкой оставшегося расстояния до цели.

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

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

7. В чем заключается основная задача искусственного интеллекта при выборе ходов в шахматах?

Многие задачи в искусственном интеллекте, будь то шахматы, планирование маршрута или доказательство математических теорем, сводятся к поиску эффективного пути в огромном пространстве возможных решений. ИИ должен найти способ эффективно исследовать это пространство, балансируя между доступными ресурсами и желаемым качеством решения, что делает его уникальным и незаменимым инструментом в современном мире.

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

Основным методом решения задач поиска являются алгоритмы поиска по дереву. Суть метода заключается в автономном моделировании процесса исследования пространства состояний путём генерации преемников уже изученных состояний, что известно как расширение состояний.

9. Каковы основные элементы задачи поиска маршрута по карте?

Задача формулируется с помощью состояний и действий: города указывают нашу текущую позицию, а действия - на способы изменения этого положения.

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

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

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

Исходное состояние дерева - стартовая точка.

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

Листовые узлы - дочерние узлы, в которые можно попасть, совершив одно действие из текущего узла.

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

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

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

Сибиу, Тимишоара, Зеринд.

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

Достижение целевого узла это и есть целевое состояние.

16. 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 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.

17. 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.

18. 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.

19. 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.

20. 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 спуститься по дереву. В некоторых случаях это значение может быть бесконечным.

21. 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.

22. Какие методы необходимо переопределить при наследовании класса Problem?

Методы actions, result, is_goal, action_cost и h.

23. Что делает метод is_goal в классе Problem ?

Метод is_goal проверяет, достигнуто ли целевое состояние.

24. Для чего используется метод action_cost в классе Problem ?

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

25. Какую задачу выполняет класс Node в алгоритмах поиска?

Node представляет узел в дереве поиска.

26. 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).

27. Что представляет собой специальный узел failure ?

Обозначение неудачи в поиске.

28. Для чего используется функция expand в коде?

Расширяет узел, генерируя дочерние узлы.

29. Какая последовательность действий генерируется с помощью функции path_actions ?

Та, с помощью которой можно добраться до узла node.

30. Чем отличается функция path_states от функции path_actions ?

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

31. 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.

32. 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.

33. Как работает метод add в классе PriorityQueue ?

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

34. В каких ситуациях применяются очереди с приоритетом?

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

35. 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 классами и объектами при написании программ с помощью языка программирования Python версии 3.x.