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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Искусственный интеллект в профессиональной сфере»
Вариант 9

Выполнил:
Иващенко Олег Андреевич
3 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и вычислительные машины»,
направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем»

_____
(подпись)

Руководитель практики:

_____
(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:

Для выполнения работы сначала была выбрана локация, где будет происходить построение маршрутов. В качестве области исследования использовалась карта Тасмании, включающая города и дороги между ними. Начальной точкой поиска был выбран город Гарденс, а целевым пунктом – город Мойна. На основе этих данных была сформирована схема маршрутов, представляющая собой граф, где вершины обозначают города, а рёбра – дороги с указанием расстояний между пунктами (рисунок 1).

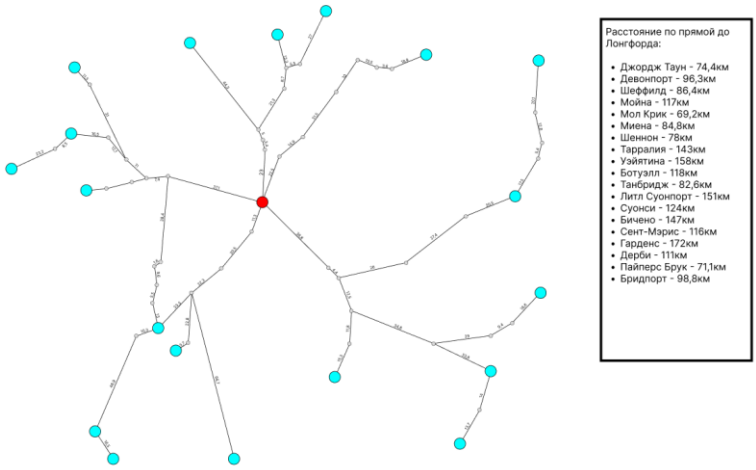

Рисунок 1 – Схема выбранных маршрутов

Для удобства восприятия дополнительно была создана визуализация маршрутов с картой на форе (рисунок 2).

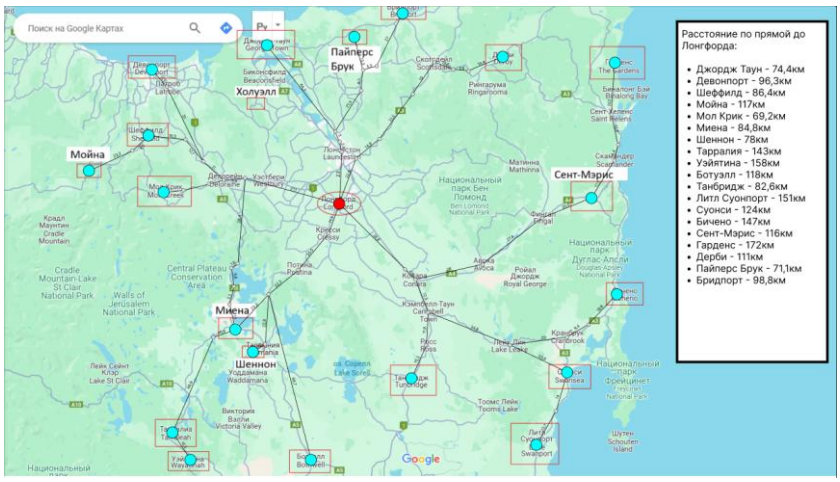

Рисунок 2 – Схема выбранным маршрутов с картой на фоне

Поиск маршрутов между указанными городами осуществлялся методом полного перебора. Это предполагало генерацию всех возможных вариантов маршрутов от Гарденс до Мойна с последующим расчётом из длины. Основной целью являлось определение кратчайшего пути. Программа вычисляла длины всех маршрутов, используя симметричную структуру графа, чтобы учесть возможность движения в обе стороны между городами.

Для визуализации итогов работы был построен граф маршрутов (рисунок 3), на котором отображались все возможные пути из Гарденс в Мойна. Все найденные маршруты выделены серыми линиями, а самый короткий маршрут обозначен красным цветом (рисунок 4). Дополнительно альтернативные пути, не являющиеся кратчайшими, но приводящие также к целевому городу, были отмечены синим цветом.

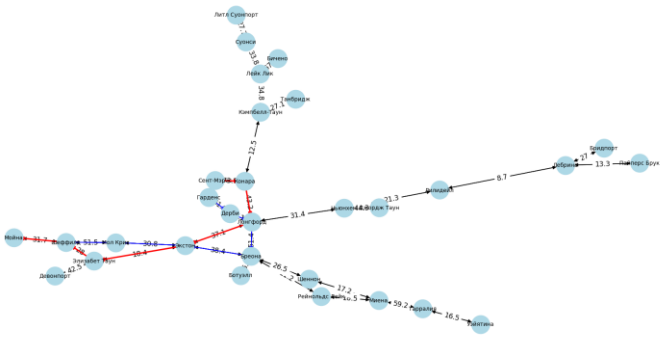

Рисунок 3 - Итоговый граф с определёнными маршрутами

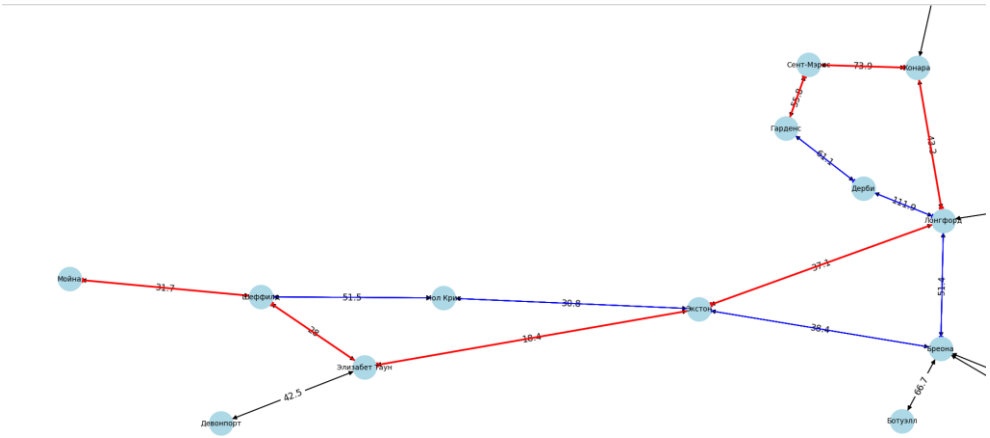

Рисунок 4 - Определённые маршруты (с приближением)

```
Все маршруты из Гарденс в Мойна :
Маршрут: Гарденс -> Сент-Мэрис -> Конара -> Лонгфорд -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 288.1 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 288.2 км
Маршрут: Гарденс -> Сент-Мэрис -> Конара -> Лонгфорд -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 324.0 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 324.1 км
Маршрут: Гарденс -> Сент-Мэрис -> Конара -> Лонгфорд -> Бреона -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 340.8 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Бреона -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 340.9 км
Маршрут: Гарденс -> Сент-Мэрис -> Конара -> Лонгфорд -> Бреона -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 376.7 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Бреона -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 376.8 км

Самый короткий маршрут: Гарденс -> Сент-Мэрис -> Конара -> Лонгфорд -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 288.1 км
```

Рисунок 5 - Вывод маршрутов в консоли

Ответы на контрольные вопросы:

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

Этот метод можно представить как попытку найти выход из затемнённого 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 неизвестных 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 в процессе поиска играет роль оценки, которая приближённо предсказывает стоимость достижения цели из текущей точки. Она помогает алгоритму быстрее находить решение, сокращая количество проверяемых вариантов, особенно в алгоритме A*.

4. Приведите пример применения эвристического поиска в реальной задаче.

Шахматы представляют собой классическую арену, где можно применить эти стратегии поиска. На первый взгляд, задача кажется простой: программа должна уметь генерировать возможные ходы и следовать базовым правилам игры.

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

Чем глубже мы погружаемся в задачу, тем сложнее становится её выполнение. Разработка ИИ для игры в шахматы требует глубокого понимания не только правил, но и бесчисленных стратегий и тактик, которые и делают шахматы настолько захватывающими.

6. Какие факторы ограничивают создание идеального шахматного 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.

Однако 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.

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

Даже если технически возможно построить структуру данных для представления всех возможных ходов в шахматах, встаёт 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 ограничено делает задачу создания эффективного шахматного ИИ особенно сложной.

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

В мире ИИ часто приходится находить баланс между скоростью вычислений, объёмом используемой памяти и способностью находить оптимальные решения. 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. Каковы основные элементы задачи поиска маршрута по карте?

Основные элементы задачи поиска маршрута по карте в контексте искусственного интеллекта включают графовое представление карты, где вершины графа – это точки интереса (перекрёстки, остановки, здания, города

или другие ключевые объекты), а рёбра графа – дороги, тропы, реки или любые другие иные пути, соединяющие вершины. Рёбра могут иметь вес, отражающий расстояние, время в пути, стоимость или сложность.

Начальная точка – это место, с которого начинается маршрут. Целевая точка – это место, куда нужно добраться. Иногда возможны дополнительные цели (множество точек).

Кратчайший путь – минимальное расстояние между двумя точками. Минимальное время – оптимизация по времени с учётом скорости движения и возможных задержек. Соответствующим образом это работает ко всех иным критериям подобного рода (будто то стоимость, сложность или что-либо ещё).

Ограничения также являются основными элементами поиска. Могут быть как физические ограничения (закрытые дороги, одностороннее движение, максимальная допустимая нагрузка), так и временные (работа мостов, временные блокировки) или иного рода.

Могут быть использованы различные алгоритмы поиска. К примеру, жадные алгоритмы используют эвристики для быстрого нахождения решений; алгоритм Дейкстры находит кратчайший путь с равными весами рёбер; A* (A-Star) – оптимальный алгоритм, использующий эвристическую оценку для ускорения поиска.

Эвристические функции, например, евклидово расстояние или манхэттенская метрика, используют для оценки расстояния до цели.

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

Оптимальность решения в данном случае предполагает нахождение маршрута с минимальной стоимостью. Среди менее эффективных вариантов могут быть маршруты через Тимишоару и Лугож с нелогичными блужданиями по кругу. Важно, чтобы итоговый маршрут завершался в Бухаресте, даже если он включает неоптимальные повторения. Оптимальное

решение – это такое, которое достигается с минимальными затратами времени и расстояния.

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

Исходное состояние дерева представляет собой стартовую точку. В нашем случае – город Арад. Это состояние символически выделяется цветом на карте. Перед нами предстоит раскрыть огромное дерево возможных путей, которое мы будем формировать постепенно, расширяя его узел за узлом. По алгоритму поиска по дереву первым шагом будет выбор листового узла для дальнейшего расширения. Так как в нашем дереве пока только один узел – Арад, мы и выбираем его.

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

Лист, листовой или терминальный узел – это узел, не имеющий дочерних элементов. Иначе говоря, это крайняя точка ветви дерева, из которой дальше нет возможности добраться до какого-либо узла.

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

На этапе расширения узла в дереве поиска происходит процесс, в котором узел, выбранный для обработки, анализируется и получаются его дочерние узлы. Это включает в себя:

- Генерация потомков – на основе состояния текущего узла создаются дочерние узлы. Обычно это происходит путём применения отдельных операций или правил, которые могут изменить состояние.
- Анализ состояния – каждое новое состояние проверяется на соответствие условиям целевой задачи. Если состояние является целевым, то процесс может быть завершён.

- Добавление в очередь – дочерние узлы добавляются в структуру данных для дальнейшей обработки (например, в очередь или стек), учитывая порядок, которым будет осуществляться их обработка (глубинный или ширинный поиск).
- Обновление информации – в зависимости от конкретной реализации может происходить обновление информации о состоянии узлов, например, вычисление значений оценки для алгоритмов вроде A*.

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

Совершив одно действие из Арада можно посетить 3 города: Сибиу, Тимишоара и Зеринд.

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

В алгоритмах поиска по дереву целевое состояние определяется в зависимости от конкретной задачи, которую необходимо решить. Основная идея заключается в том, чтобы установить условия, при которых поиск может завершиться, и мы получаем искомое решение. Целевое состояние определяется следующим образом:

- Определение условий успеха – целевое состояние задаётся набором критериев, которые должны быть выполнены для того, чтобы можно было считать задачу решённой. Это может быть, например, нахождение определённого узла в графе или достижение заданной конфигурации.
- Эволюция состояния. Каждое состояние в дереве является результатом применения определённых операций или действий над предыдущими состояниями. Определение целевого состояния связано с тем, как эти операции меняют текущее состояние и приводят его к искомому.
- Параметры и свойства – целевое состояние может быть сформулировано через определённые параметры и свойства, которые должны

быть выполнены. Например, в задачах поиска пути целевым может быть узел назначения, а в задачах логического вывода – формула, которая должна быть доказана.

- Использование состояний – в некоторых алгоритмах, таких как поиск в глубину или ширину, каждое состояние проверяется на соответствие целевому, и при попадании в целевое состояние осуществляется возврат результата.
- Хранение информации – для поиска целевых состояний может быть полезно использовать хранилище для уже посещённых состояний, чтобы избежать повторного посещения и избыточных вычислений.

16. Какие основные шаги выполняет алгоритм поиска по дереву?

Алгоритм действует следующим образом: если нет кандидатов для расширения, алгоритм возвращает неудачу; в противном случае выбирается листовой узел для расширения в соответствии со стратегией, определяющей, какой из листовых узлов будет расширен; если узел содержит целевое состояние, возвращается соответствующее решение – путь в дереве, приведший к этому листовому узлу с целевым состоянием; если нет, узел расширяется и полученные узлы добавляются к дереву. Этот процесс и называется поиском по дереву.

17. Чем различаются состояния и узлы в дереве поиска?

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

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

Функция преемника задаёт множество возможных состояний, которые можно достичь из данного состояния при выполнении различных действий.

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

Временная и пространственная сложность алгоритмов характеризуются тремя ключевыми значениями:

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.

Таким образом, 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?

Качество алгоритма оценивается по четырём характеристикам: полноте, временной сложности, пространственной сложности и 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.

21. Какую роль выполняет класс ***Problem*** в приведенном коде?

Класс Problem – абстрактный класс для формальной задачи. Новый домен специализирует это класс, переопределяя *actions* и *results*, и, возможно, другие методы. Эвристика по умолчанию равна 0, а стоимость действия по умолчанию равна 1 для всех состояний. Когда создаётся экземпляр подкласса, нужно указать «начальное» и «целевое» состояние (или задать метод *is_goal*) и, возможно, другие ключевые слова для подкласса.

Класс служим 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***?

При наследовании класса Problem необходимо переопределить методы:

- actions(self, state);
- results(self, state, action);
- is_goal(self, state);
- action_cost(self, s, a, s1);
- h(self, node).

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***?

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

25. Какую задачу выполняет класс ***Node*** в алгоритмах поиска?

Объект, создаваемый при помощи класса Node, является узлом в дереве поиска.

26. Какие параметры принимает конструктор класса ***Node***?

Класс Node принимает следующие параметры:

- *state* – текущее состояние;
- *parent* – ссылка на родительский узел;
- *action* – действие, которое привело к этому узлу;
- *path_cost* – стоимость пути.

27. Что представляет собой специальный узел ***failure***?

Специальный узел *failure* нужен для обозначения неудачи в поиске.

failure = Node('failure', path_cost=math.inf)

28. Для чего используется функция ***expand*** в коде?

Функция *expand* расширяет узел, генерируя дочерние узлы.

```
def expand(problem, node):
    "Раскрываем узел, создав дочерние узлы."
```

29. Какая последовательность действий генерируется с помощью функции ***path_actions***?

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

```
def path_actions(node):
    "Последовательность действий, чтобы добраться до этого узла."
```

30. Чем отличается функция ***path_states*** от функции ***path_actions***?

Отличие функций заключается в том, что функция *path_actions* возвращает последовательность **действий**, а функция *path_states* возвращает последовательность **состояний**.

31. Какой тип данных используется для реализации ***FIFOQueue***?

FIFOQueue – это очередь, где первый добавленный элемент будет первым извлечён. Она реализуется с помощью *deque* из моделя *collections*. *deque* – это обобщённая версия стека и очереди, которая поддерживает добавление и удаление элементов с обоих концов.

32. Чем отличается очередь ***FIFOQueue*** от ***LIFOQueue***?

Отличие очередей заключается в том, что в очереди FIFOQueue первым извлечённым элементом будет первый добавленный (First In First Out), а в очереди LIFOQueue первым извлечённым элементом будет последний добавленный (Last In First Out).

33. Как работает метод ***add*** в классе ***PriorityQueue***?

add – метод для добавления элемента в очередь. Каждый элемент добавляется в кучу с его приоритетом, определённым функцией *key*.

34. В каких ситуациях применяются очереди с приоритетом?

Когда нужно обработать наиболее важные элементы в первую очередь.

35. Как функция ***heappop*** помогает в реализации очереди с приоритетом?

Функция *heappop* из модуля *heapq* гарантирует извлечение элемента с наименьшим приоритетом.

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

Ссылка на репозиторий:

IUnnamedUserI/AI_1: Искусственный интеллект в профессиональной сфере. Лабораторная работа №1