

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»**

Выполнил:  
Магомедов Имран Борисович  
3 курс, группа «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А., доцент департамента  
цифровых, робототехнических систем и  
электроники

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

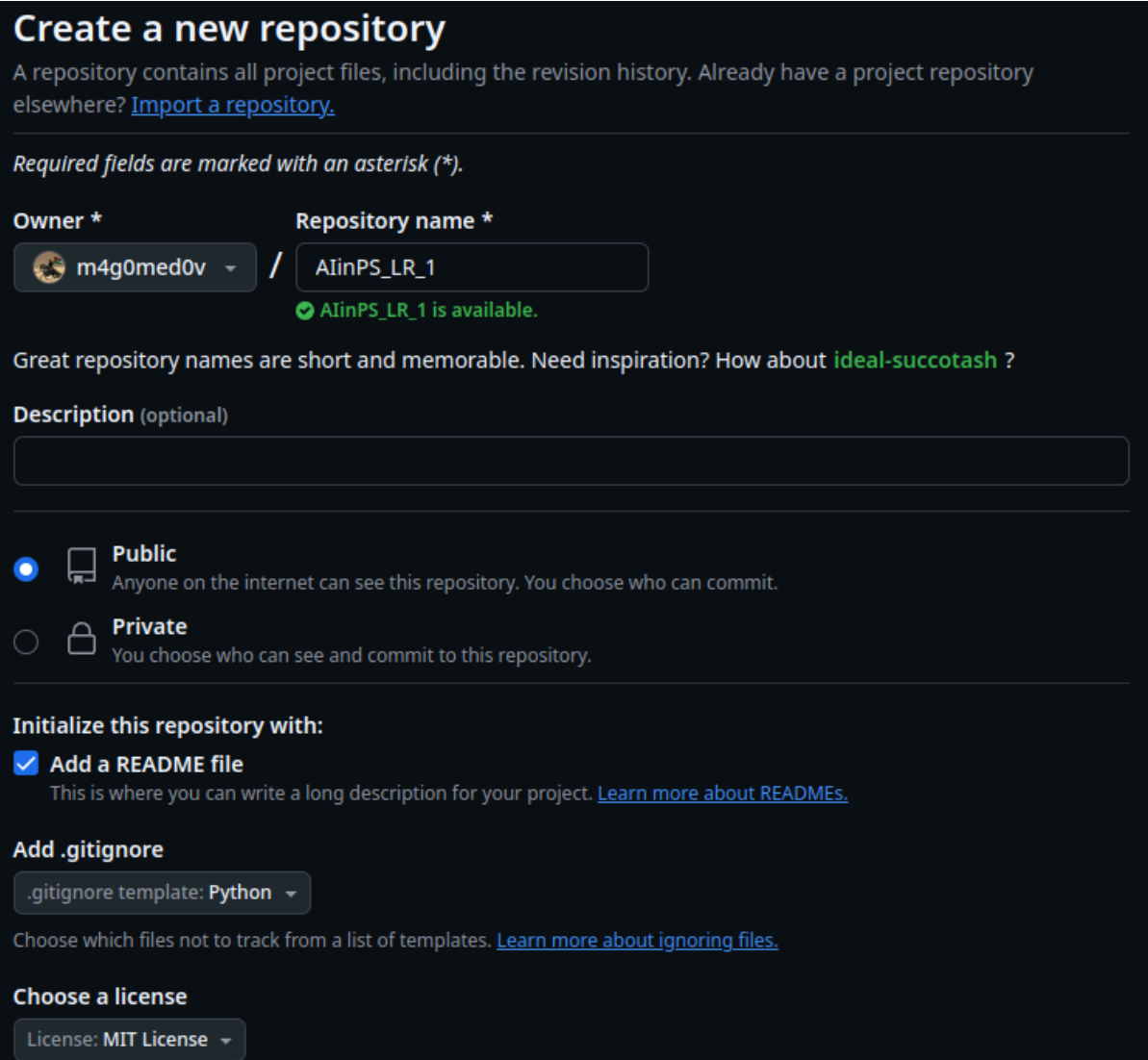
Ставрополь, 2024 г.

**Тема:** Исследование методов поиска в пространстве состояний.

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

### Методика выполнения работы

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором использовал лицензию MIT и язык программирования Python.



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, it states 'Required fields are marked with an asterisk (\*)'. The 'Owner' field is set to 'm4g0med0v' and the 'Repository name' field is 'AIinPS\_LR\_1', with a green checkmark indicating it is available. There is a suggestion for 'ideal-succotash'. The 'Description' field is optional and empty. Under 'Visibility', 'Public' is selected. The 'Initialize this repository with:' section has 'Add a README file' checked. The '.gitignore' section has 'Python' selected as the template. Finally, the 'Choose a license' section has 'MIT License' selected.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* / Repository name \*

m4g0med0v / AIinPS\_LR\_1

✓ AIinPS\_LR\_1 is available.

Great repository names are short and memorable. Need inspiration? How about **ideal-succotash** ?

Description (optional)

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License

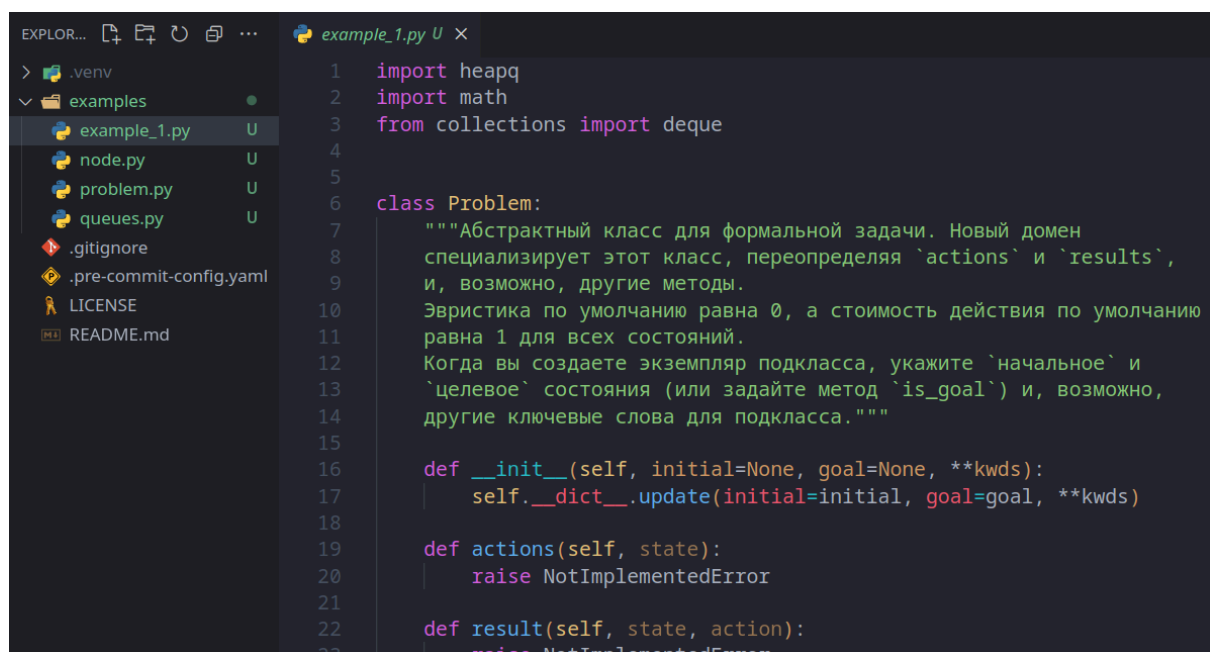
3. Выполнил клонирования созданного репозитория.

```

~ /work git clone https://github.com/m4g0med0v/AIinPS_LR_1.git
Клонирование в «AIinPS_LR_1»...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Получение объектов: 100% (5/5), готово.

```

4. Проработал примеры лабораторной работы. Создал для каждого примера отдельный модуль языка Python. Зафиксировал изменения в репозитории.



```

1 import heapq
2 import math
3 from collections import deque
4
5
6 class Problem:
7     """Абстрактный класс для формальной задачи. Новый домен
8     специализирует этот класс, переопределяя `actions` и `results`,
9     и, возможно, другие методы.
10    Эвристика по умолчанию равна 0, а стоимость действия по умолчанию
11    равна 1 для всех состояний.
12    Когда вы создаете экземпляр подкласса, укажите `начальное` и
13    `целевое` состояния (или задайте метод `is_goal`) и, возможно,
14    другие ключевые слова для подкласса."""
15
16    def __init__(self, initial=None, goal=None, **kwargs):
17        self.__dict__.update(initial=initial, goal=goal, **kwargs)
18
19    def actions(self, state):
20        raise NotImplementedError
21
22    def result(self, state, action):
23        raise NotImplementedError

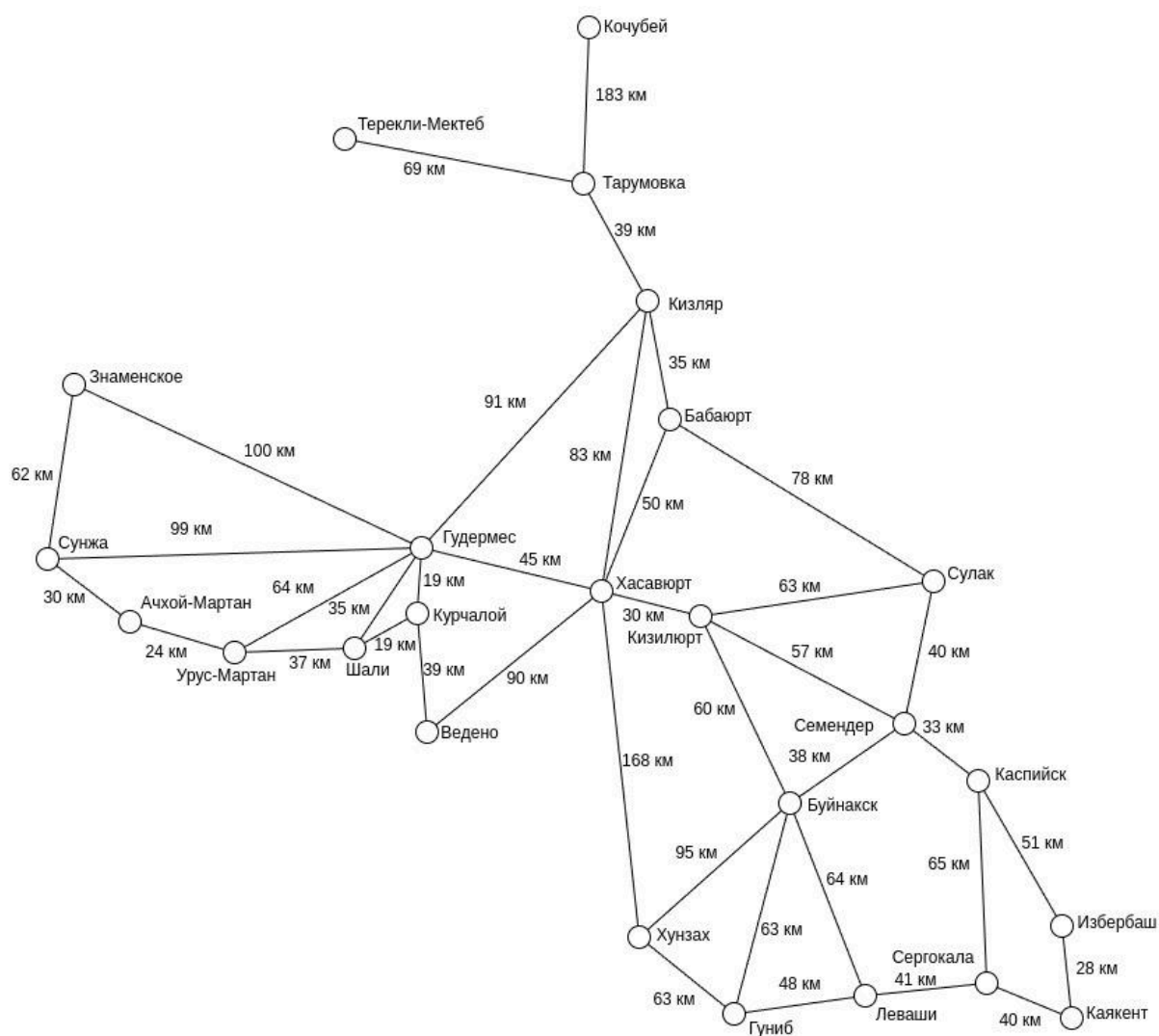
```

```

~ /w/AIinPS_LR_1 main +1 +4 git commit -m 'added examples from LR '
check yaml.....(no files to check)Skipped
fix end of files.....Passed
trim trailing whitespace.....Passed
ruff.....Passed
ruff-format.....Passed
[main f931ad1] added examples from LR
4 files changed, 223 insertions(+)
create mode 100644 examples/example_1.py
create mode 100644 examples/node.py
create mode 100644 examples/problem.py
create mode 100644 examples/queues.py

```

5. Воспользовавшись сервисом Google Maps создал граф из 20 и более населенных пунктов.



6. Определил минимальный маршрут между “Гудермес” и “Леваши”, который проходит через три промежуточных населенных пункта. Короткий путь: Гудермес - Хасавюрт - Кизилюрт - Буйнакск - Леваши = 189 км.



```

27
28 # Функция решения задачи коммивояжёра
29 def solve_tsp(graph, start_node):
30     # Получаем список всех городов, кроме начального
31     cities = list(graph.keys())
32     cities.remove(start_node)
33
34     shortest_path = None
35     min_distance = float("inf")
36
37     # Проходим по всем перестановкам городов
38     for perm in permutations(cities):
39         path = [start_node] + list(perm)
40         distance = calculate_path_distance(path, graph)
41         if distance < min_distance:
42             min_distance = distance
43             shortest_path = path
44
45     return shortest_path, min_distance
46
47
48 # Решение задачи коммивояжёра для заданного графа с началом в "Гудермес"
49 start_time = time()
50 solution_path, solution_distance = solve_tsp(graph_12_points, "Гудермес")
51 end_time = time()

```

Создал еще один файл с графами различным количеством населенных пунктов.

```

graphs.py ×
1 # Основной граф
2 graph_26_points = {
3     "Гудермес": {
4         "Знаменское": 100,
5         "Сунжа": 99,
6         "Урус-Мартан": 64,
7         "Шали": 35,
8         "Курчалой": 19,
9         "Хасавюрт": 45,
10        "Кизляр": 91,
11    },
12    "Знаменское": {"Гудермес": 100, "Сунжа": 62},
13    "Сунжа": {"Знаменское": 62, "Гудермес": 99, "Ачхой-Мартан": 30},
14    "Ачхой-Мартан": {"Сунжа": 30, "Урус-Мартан": 24},
15    "Урус-Мартан": {"Ачхой-Мартан": 24, "Гудермес": 64, "Шали": 37},
16    "Шали": {"Гудермес": 35, "Курчалой": 19, "Урус-Мартан": 37},
17    "Курчалой": {"Ведено": 39, "Гудермес": 19, "Шали": 19},
18    "Ведено": {"Курчалой": 39, "Хасавюрт": 90},
19    "Хасавюрт": {
20        "Гудермес": 45,
21        "Кизляр": 83,
22        "Бабаюрт": 50,

```

Для начала используем граф с 7 пунктами.

```
~/w/AIinPS_LR_1 main python src/travelling\ salesman.py
Оптимальный маршрут: ['Гудермес', 'Шали', 'Курчалой', 'Ведено', 'Буйнакск', 'Леваши', 'Хасавюрт']
Общая длина пути: 422 км
Время затраченное на поиск: 0.0011518001556396484
```

Для 10 пунктов.

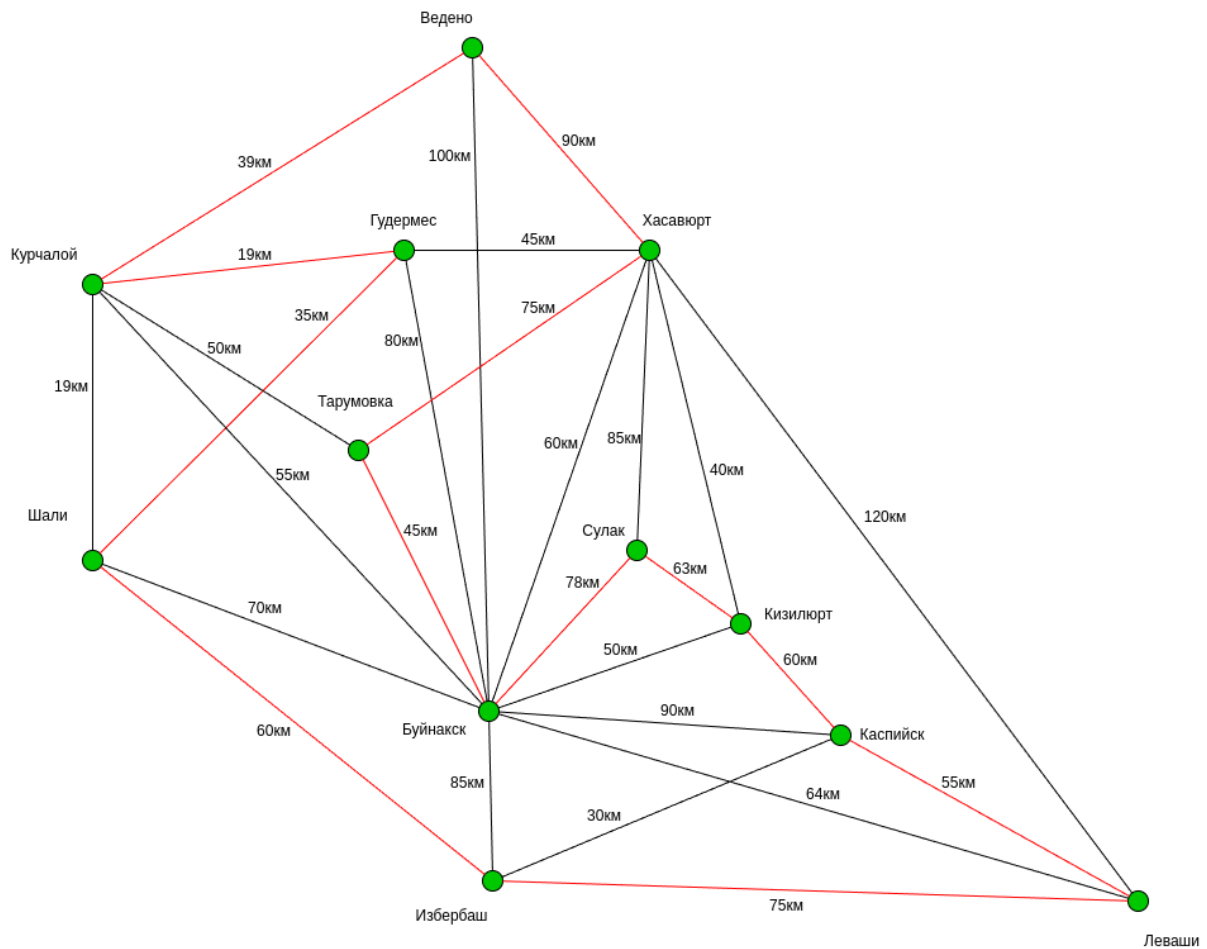
```
~/w/AIinPS_LR_1 main !1 python src/travelling\ salesman.py
Оптимальный маршрут: ['Гудермес', 'Шали', 'Избербаш', 'Каспийск', 'Леваши', 'Буйнакск', 'Кизилюрт', 'Хасавюрт', 'Ведено', 'Курчалой']
Общая длина пути: 482 км
Время затраченное на поиск: 0.7620580196380615
```

Для 12 пунктов.

```
~/w/AIinPS_LR_1 main !1 python src/travelling\ salesman.py
Оптимальный маршрут: ['Гудермес', 'Курчалой', 'Ведено', 'Хасавюрт', 'Тарумовка', 'Буйнакск', 'Сулак', 'Кизилюрт', 'Каспийск', 'Леваши', 'Избербаш', 'Шали']
Общая длина пути: 694 км
Время затраченное на поиск: 93.00257515907288
```

Дальше не стал пробовать т.к. занимает более 10 минут времени

Получившийся граф:



8. Зафиксировал сделанные изменения в репозитории.

```

~/w/AIinPS_LR_1 main !2 git add .
~/w/AIinPS_LR_1 main +1 git commit -m "modified graphs"
check yml.....(no files to check) Skipped
fix end of files..... Passed
trim trailing whitespace..... Passed
ruff..... Passed
ruff-format..... Passed
[main 278ce92] modified graphs
1 file changed, 9 insertions(+), 1 deletion(-)

```

9. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.

10. Отправил сделанные изменения на сервер GitHub.

## Контрольные вопросы



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

**Метод "слепого поиска"** — это базовый метод поиска в ИИ, который не использует информацию о целевом состоянии для выбора следующего шага. Он работает по принципу проб и ошибок, исследуя возможные решения без учета их близости к цели.

2. Как отличается эвристический поиск от слепого поиска?

**Эвристический поиск** отличается от слепого тем, что использует дополнительную информацию для оценки вероятности достижения цели от текущего состояния. Это позволяет более эффективно сокращать пространство поиска, например, с использованием алгоритма  $A^*$ .

3. Какую роль играет эвристика в процессе поиска?

**Эвристика** в процессе поиска играет роль оценки, которая помогает алгоритму выбирать более перспективные пути к цели, снижая количество рассматриваемых вариантов и ускоряя поиск.

4. Приведите пример применения эвристического поиска в реальной задаче.

**Пример эвристического поиска** — алгоритм  $A^*$  для поиска кратчайшего пути на карте. Он используется в задачах маршрутизации, например, в навигационных системах, где нужно найти оптимальный маршрут между двумя городами с учетом расстояний.

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

**Полное исследование всех возможных ходов в шахматах** затруднительно из-за огромного количества комбинаций, которые увеличиваются экспоненциально с каждым ходом. Это делает невозможным рассмотрение всех возможных вариантов даже для мощных компьютеров.

6. Какие факторы ограничивают создание идеального шахматного ИИ?

**Факторы, ограничивающие создание идеального шахматного ИИ:** ограниченные вычислительные мощности, сложность оценивания позиций на доске и необходимость быстрого принятия решений в условиях ограничения времени на ход.

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

**Основная задача ИИ при выборе ходов в шахматах** — это нахождение оптимального хода, который максимизирует вероятность победы, учитывая текущую позицию и предсказания будущих ходов противника.

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

**Алгоритмы ИИ балансируют между скоростью вычислений и оптимальностью решений**, применяя эвристики и стратегии отсечения, чтобы быстро отбрасывать менее перспективные варианты и сосредоточиться на более вероятных путях к успеху.

9. Каковы основные элементы задачи поиска маршрута по карте?

**Основные элементы задачи поиска маршрута по карте** включают начальное состояние (город, с которого начинается маршрут), целевое состояние (город назначения), доступные действия (переходы между городами), и функцию стоимости (расстояние между городами).

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

**Оптимальность решения задачи маршрутизации на карте Румынии** оценивается на основе минимизации общей стоимости (например, расстояния или времени), необходимой для достижения цели от начальной точки.

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

**Исходное состояние дерева поиска** в задаче маршрутизации по карте Румынии — это начальный город, с которого начинается поиск, например, город Арад.

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

**Листовые узлы** — это узлы, которые не имеют дочерних узлов. Они представляют собой состояния, которые еще не были расширены в рамках поиска.

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

**На этапе расширения узла** в дереве поиска из текущего узла генерируются его дочерние узлы, представляющие возможные переходы в новые состояния.

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

**Города, которые можно посетить из Арада** в задаче поиска по карте Румынии, включают Сибиу, Тимишоару и Зеринд.

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

**Целевое состояние** в алгоритме поиска по дереву определяется как состояние, которое соответствует цели задачи, например, достижение города Бухарест.

16. Какие основные шаги выполняет алгоритм поиска по дереву?

**Основные шаги алгоритма поиска по дереву** включают инициализацию начального узла, проверку на достижение цели, расширение узлов и добавление новых узлов в очередь до достижения целевого состояния или исчерпания вариантов.

17. Чем различаются состояния и узлы в дереве поиска?

**Состояния** — это различные конфигурации задачи, в то время как **узлы** содержат информацию о состоянии, включая родителя, действие, которое привело к этому состоянию, и стоимость пути до него.

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

**Функция преемника** генерирует возможные состояния, которые можно достичь из текущего состояния, и используется для расширения узлов в алгоритме поиска.

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

**Параметры  $b$  (разветвление),  $d$  (глубина решения) и  $m$  (максимальная глубина)** влияют на время и пространственную сложность поиска, определяя количество узлов, которые будут сгенерированы и расширены в процессе поиска.

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

**Оценка алгоритмов поиска по дереву** проводится по критериям полноты (нахождение решения, если оно существует), временной сложности (количество сгенерированных узлов), пространственной сложности (память, необходимая для хранения узлов), и оптимальности (нахождение наилучшего решения).

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

**Класс `Problem`** в коде служит шаблоном для описания конкретных задач, предоставляя методы для определения доступных действий, проверки достижения цели и вычисления стоимости действий.

22. Какие методы необходимо переопределить при наследовании класса `Problem` ?

**Методы, которые нужно переопределить в классе Problem:** actions, result, is\_goal, а также, возможно, action\_cost и h (эвристическая оценка) для конкретных задач.

23. Что делает метод is\_goal в классе Problem ?

**Метод is\_goal** проверяет, достигнуто ли целевое состояние для текущего состояния задачи.

24. Для чего используется метод action\_cost в классе Problem ?

**Метод action\_cost** используется для вычисления стоимости перехода между состояниями, учитывая выполненное действие.

25. Какую задачу выполняет класс Node в алгоритмах поиска?

**Класс Node** в алгоритмах поиска представляет узел дерева поиска, хранящий текущее состояние, ссылку на родителя, выполненное действие и накопленную стоимость пути.

26. Какие параметры принимает конструктор класса Node ?

**Конструктор класса Node** принимает параметры: текущее состояние, родительский узел, действие, приведшее к этому узлу, и стоимость пути до узла.

27. Что представляет собой специальный узел failure ?

**Специальный узел failure** представляет неудачу в поиске, указывая, что не найдено решение задачи.

28. Для чего используется функция expand в коде?

**Функция expand** используется для генерации дочерних узлов, применяя доступные действия к текущему состоянию и создавая новые узлы.

29. Какая последовательность действий генерируется с помощью функции path\_actions ?

**Функция path\_actions** возвращает последовательность действий, которые привели к данному узлу, начиная с корневого узла.

30. Чем отличается функция path\_states от функции path\_actions ?

**Функция `path_states`** возвращает последовательность состояний, пройденных для достижения текущего узла, в отличие от `path_actions`, которая возвращает сами действия.

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

**Для реализации `FIFOQueue`** обычно используется `deque` из модуля `collections` в Python.

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

**`FIFOQueue` отличается от `LIFOQueue`** тем, что в `FIFOQueue` первым извлекается элемент, который был добавлен первым, а в `LIFOQueue` — последний добавленный элемент (то есть поведение стека).

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

**Метод `add` в классе `PriorityQueue`** добавляет элемент в очередь с приоритетом, используя функцию `heappush`, чтобы поддерживать упорядоченность очереди.

34. В каких ситуациях применяются очереди с приоритетом?

**Очереди с приоритетом** применяются, когда необходимо обработать элементы с разными уровнями важности, например, в алгоритме  $A^*$  для обработки узлов с наименьшей стоимостью пути первым.

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

**Функция `heappop`** помогает в реализации очереди с приоритетом, извлекая элемент с минимальным приоритетом, что делает процесс извлечения наиболее приоритетных элементов более эффективным.