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

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Искусственный интеллект в профессиональной сфере»

Выполнила:
Гайчук Дарья Дмитриевна
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)

Руководитель практики:
Воронкин Р.А.-доцент департамента
цифровых, роботехнических систем и
электроники института перспективной
инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: «Исследование поиска с ограничением глубины»

Цель работы: приобретение навыков по работе с поиском с ограничением глубины с помощью языка программирования Python версии 3.x

Ссылка на репозиторий: https://github.com/Ichizuchi/PR_AI_004

Порядок выполнения работы:

1. Создала новый репозиторий и клонировала его на свой компьютер.

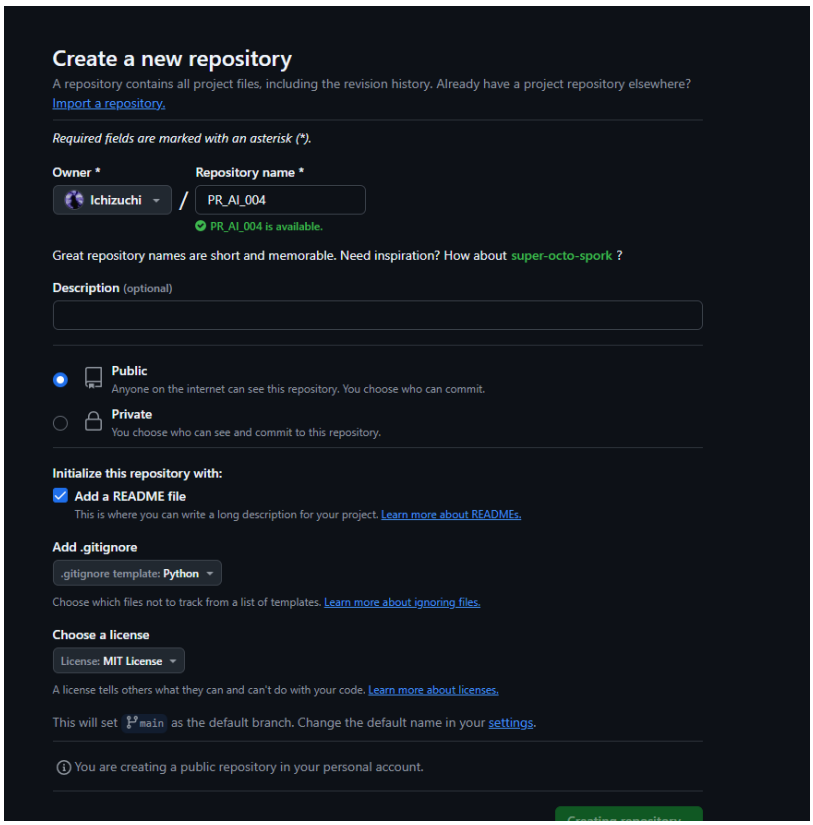


Рисунок 1. Создание репозитория

2. Клонировала репозиторий на свой компьютер.

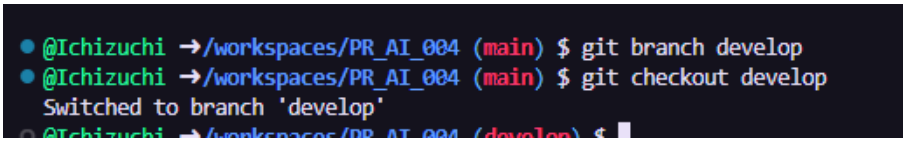


Рисунок 2. Модель ветвления git-flow

3. Создала виртуальное окружение Anaconda с именем репозитория.

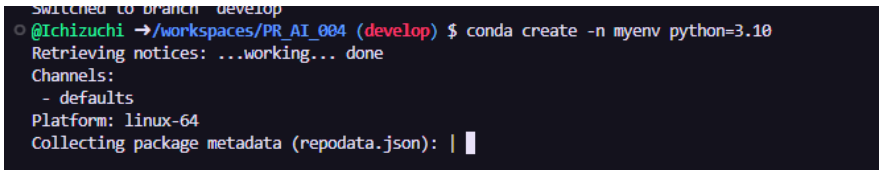


Рисунок 3. Создание виртуального окружения

Вариант 3

Задание №1. **Система навигации робота пылесоса:** дано дерево, где каждый узел представляет собой комнату в доме. Узлы связаны в соответствии с возможностью перемещения робота из одной комнаты в другую. Необходимо определить, существует ли путь от начальной комнаты (корень дерева) к целевой комнате (узел с заданным значением), так, чтобы робот не превысил лимит по глубине перемещения.

Tasks > Task_1.py > ...

```python
class BinaryTreeNode:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def __repr__(self):
        return f"<{self.value}>"


def is_path_within_depth(node, goal, limit, depth=0):

    if node is None:
        return False

    print(f"На глубине {depth} проверяем комнату {node}")

    # Если найден целевой узел
    if node.value == goal:
        return True

    if depth >= limit:
        return False

    left_result = is_path_within_depth(node.left, goal, limit, depth + 1)
    if left_result:
        return True

    right_result = is_path_within_depth(node.right, goal, limit, depth + 1)
    return right_result

# Построение дерева
root = BinaryTreeNode(
    1,
    BinaryTreeNode(2, None, BinaryTreeNode(4)),
    BinaryTreeNode(3, BinaryTreeNode(5), None),
)

goal = 4
```

Рисунок 4. Код программы

```python
# Вывод результата
if found:
    print("Найден на глубине: True")
else:
    print("Найден на глубине: False")
```

ПРОБЛЕМЫ 1 ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ

```
/home/codespace/.python/current/bin/python3 /workspaces/PR_AI_004/Tasks/Task_1.py
@Ichizuchi →/workspaces/PR_AI_004 (develop) $ /home/codespace/.python/current/bin/pyth
На глубине 0 проверяем комнату <1>
На глубине 1 проверяем комнату <2>
На глубине 2 проверяем комнату <4>
Найден на глубине: True
```

Рисунок 6. Вывод задания №1

Задание №2. **Система управления складом:** найти наименее затратный путь к товару, ограничив поиск заданной глубиной, чтобы гарантировать, что поиск займет приемлемое время.

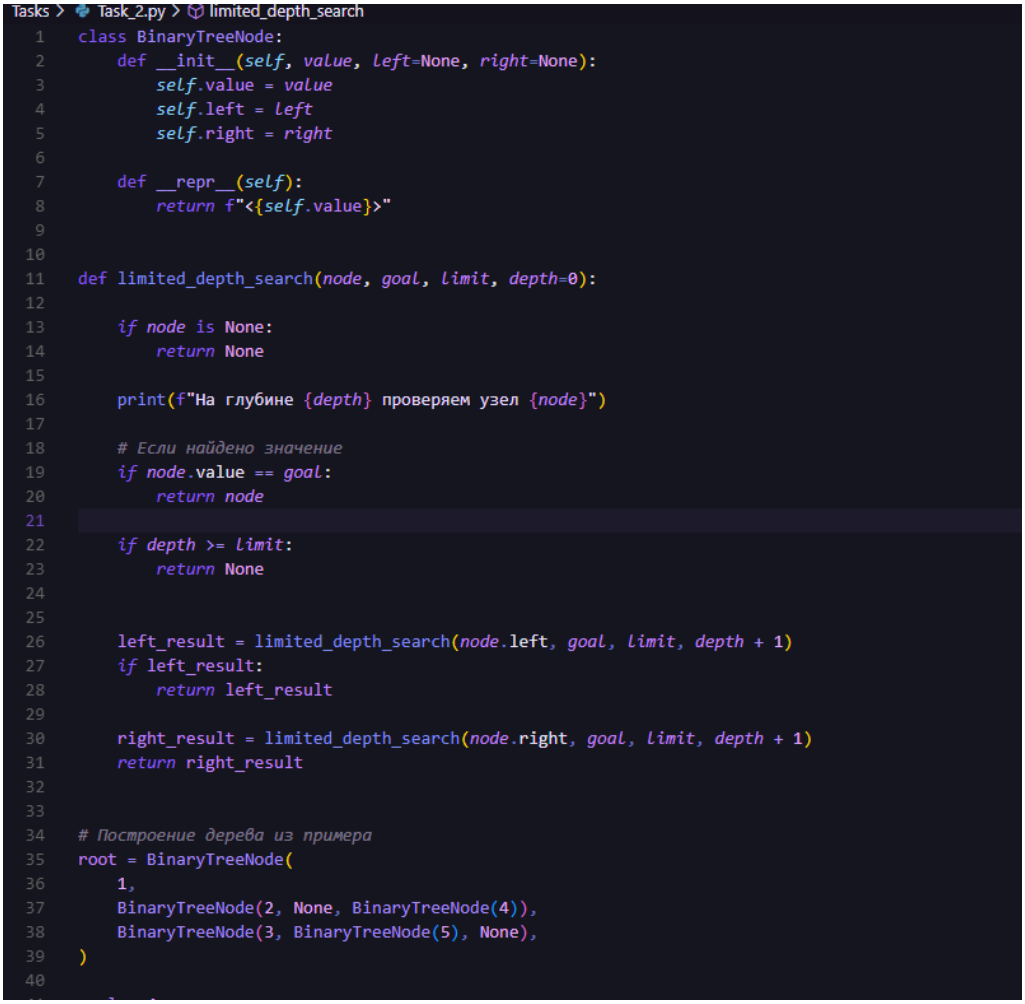


Рисунок 7. Код программы

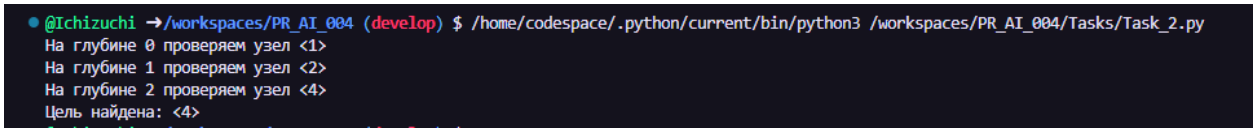
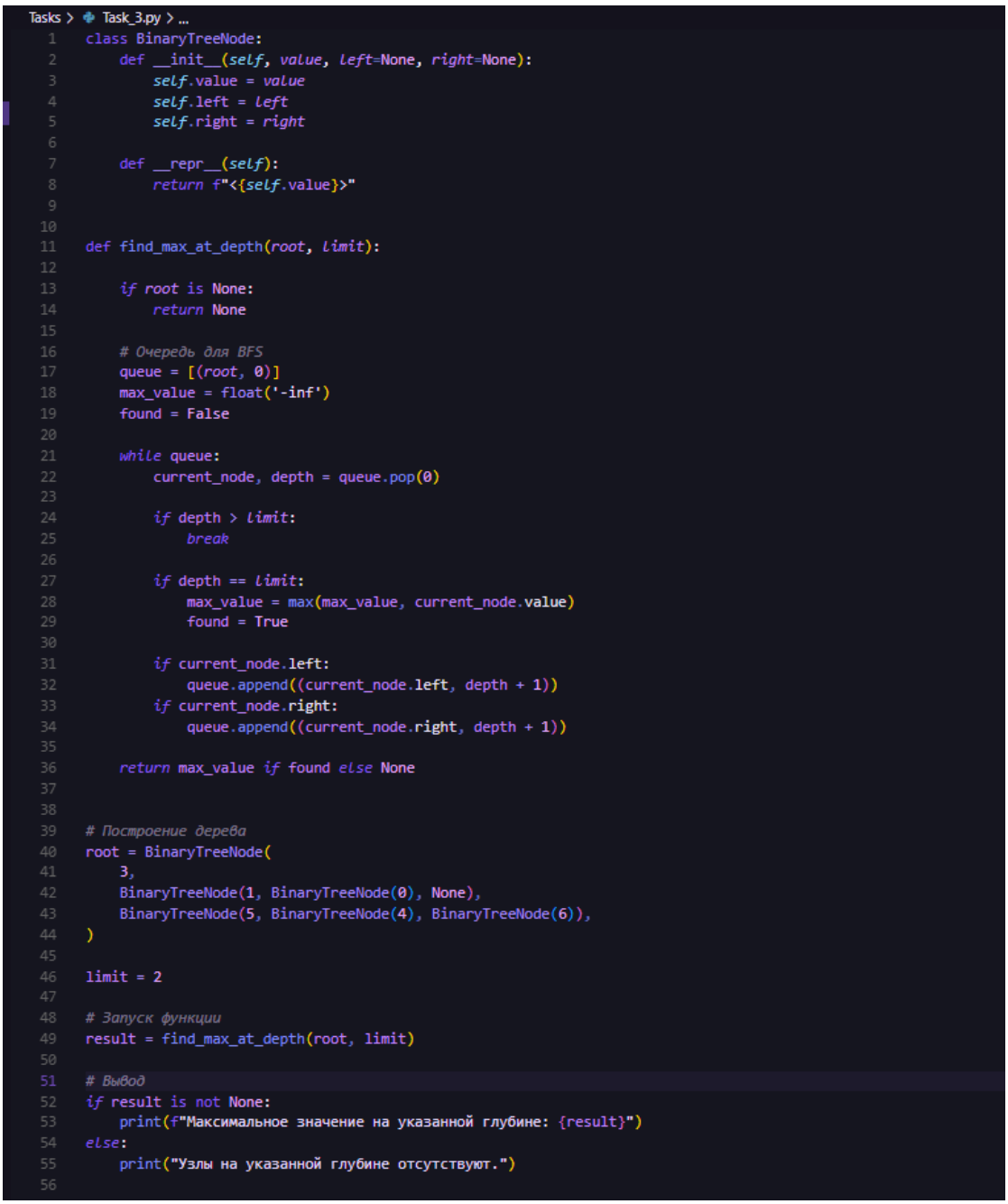


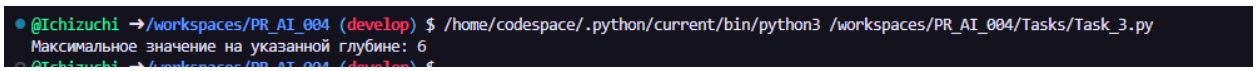
Рисунок 9. Вывод задания №2

Задание №3. Система автоматического управления инвестициями: Цель состоит в том, чтобы найти наилучший исход (максимальную прибыль) на определённой глубине принятия решений, учитывая ограниченные ресурсы и время на анализ.



```
Tasks > Task_3.py > ...
class BinaryTreeNode:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def __repr__(self):
        return f"<{self.value}>"


def find_max_at_depth(root, limit):

    if root is None:
        return None

    # Очередь для BFS
    queue = [(root, 0)]
    max_value = float('-inf')
    found = False

    while queue:
        current_node, depth = queue.pop(0)

        if depth > limit:
            break

        if depth == limit:
            max_value = max(max_value, current_node.value)
            found = True

        if current_node.left:
            queue.append((current_node.left, depth + 1))
        if current_node.right:
            queue.append((current_node.right, depth + 1))

    return max_value if found else None


# Построение дерева
root = BinaryTreeNode(
    3,
    BinaryTreeNode(1, BinaryTreeNode(0), None),
    BinaryTreeNode(5, BinaryTreeNode(4), BinaryTreeNode(6)),
)

limit = 2

# Запуск функции
result = find_max_at_depth(root, limit)

# Вывод
if result is not None:
    print(f"Максимальное значение на указанной глубине: {result}")
else:
    print("Узлы на указанной глубине отсутствуют.")
```

Рисунок 10. Код программы



```
@Ichizuchi ➜/workspaces/PR_AI_004 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/PR_AI_004/Tasks/Task_3.py
Максимальное значение на указанной глубине: 6
```

Рисунок 12. Вывод задания №3

Задание №4. **Поиск с ограничением глубины для графов**: Использовать алгоритм из лабораторной работы 1 для поиска минимального расстояния.

Tasks > Task_4.py > ...

```python
# Поиск с ограничением глубины по личному графу
from collections import deque

# Граф с расстояниями между городами
graph = {
    "Стамбул": [("Бурса", 155), ("Эскишехир", 480), ("Анкара", 450)],
    "Бурса": [("Стамбул", 155), ("Анкара", 480), ("Маниса", 70)],
    "Эскишехир": [("Стамбул", 480), ("Анкара", 230)],
    "Анкара": [("Эскишехир", 230), ("Стамбул", 450), ("Бурса", 480), ("Сивас", 320), ("Конья", 260)],
    "Маниса": [("Бурса", 70), ("Измир", 330)],
    "Измир": [("Маниса", 330), ("Анталья", 460)],
    "Конья": [("Анкара", 260), ("Анталья", 290), ("Мерсин", 460)],
    "Анталья": [("Измир", 460), ("Конья", 290)],
    "Мерсин": [("Конья", 460), ("Адана", 70)],
    "Адана": [("Мерсин", 70), ("Газантеп", 220)],
    "Газантеп": [("Адана", 220), ("Шанлыурфа", 150)],
    "Шанлыурфа": [("Газантеп", 150), ("Диярбакыр", 180)],
    "Диярбакыр": [("Шанлыурфа", 180), ("Ван", 320)],
    "Ван": [("Диярбакыр", 320), ("Эрзурум", 370)],
    "Эрзурум": [("Ван", 370), ("Трабзон", 250)],
    "Трабзон": [("Эрзурум", 250), ("Самсун", 330)],
    "Самсун": [("Трабзон", 330), ("Сивас", 410)],
    "Сивас": [("Самсун", 410), ("Анкара", 320), ("Кайсери", 240)],
    "Кайсери": [("Сивас", 240), ("Малатья", 240)],
    "Малатья": [("Кайсери", 240), ("Шанлыурфа", 90)]
}

# Алгоритм поиска с ограничением глубины
def depth_limited_search(graph, start, goal, limit):
    def recursive_dls(node, depth):
        if node == goal:
            return (True, depth)
        if depth == limit:
            return (False, float('inf'))  # Ограничение
        min_distance = float('inf')
        for neighbor, distance in graph.get(node, []):
            found, dist = recursive_dls(neighbor, depth + 1)
            if found:
                return (True, dist + distance)
            min_distance = min(min_distance, dist)
        return (False, min_distance)

    found, result = recursive_dls(start, 0)
    return result if found else None

# Проверка
start = "Стамбул"
goal = "Анкара"
limit = 3
result = depth_limited_search(graph, start, goal, limit)
print(f"Минимальное расстояние: {result}" if result else "Решение не найдено")
```

Рисунок 13. Код программы

```
# Проверка
start = "Стамбул"
goal = "Анкара"
limit = 3
result = depth_limited_search(graph, start, goal, limit)
print(f"Минимальное расстояние: {result}" if result else "Решение не найдено")

ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ

/home/codespace/.python/current/bin/python3 /workspaces/PR_AI_004/Tasks/Task_4.py
@Ichizuchi →/workspaces/PR_AI_004 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/PR_AI_004/Tasks/Task_4.py
Минимальное расстояние: 763
```

Рисунок 14. Вывод задания №4

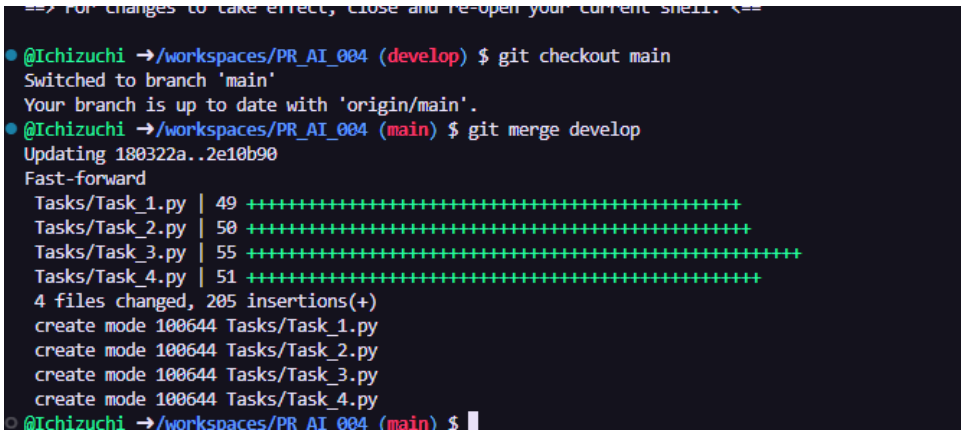


Рисунок 15. Слияние веток

Ответы на контрольные вопросы

1. Что такое поиск с ограничением глубины, и как он решает проблему бесконечных ветвей?

Поиск с ограничением глубины — это разновидность поиска в глубину, которая ограничивает максимальную глубину рекурсивного поиска. Он предотвращает углубление в бесконечные ветви, прерывая дальнейшее исследование узлов, находящихся за пределами установленной глубины.

2. Какова основная цель ограничения глубины в данном методе поиска?

Основная цель ограничения глубины — предотвратить зацикливание алгоритма на бесконечных путях, особенно если граф содержит бесконечные ветви или циклы. Это улучшает производительность и позволяет избежать переполнения стека.

3. В чем разница между поиском в глубину и поиском с ограничением глубины?

— Поиск в глубину углубляется до максимально возможной глубины (или до достижения решения), даже если это приводит к зацикливанию.

— Поиск с ограничением глубины устанавливает фиксированный предел, за который алгоритм не углубляется, даже если решение находится глубже.

4. Какую роль играет проверка глубины узла в псевдокоде поиска с ограничением глубины?

Проверка глубины узла позволяет остановить исследование, если текущий узел находится на уровне, превышающем установленный предел. Это предотвращает углубление за заданную границу.

5. Почему в случае достижения лимита глубины функция возвращает «обрезание»?

Функция возвращает "обрезание" (cutoff), чтобы сигнализировать, что ограничение глубины не позволило продолжить поиск. Это полезно для итеративного углубления, которое может затем увеличить глубину и продолжить поиск.

6. В каких случаях поиск с ограничением глубины может не найти решение, даже если оно существует?

— Если решение находится глубже установленного ограничения.

— Если структура графа делает узел недоступным из-за циклов или обрезаний.

7. Как поиск в ширину и в глубину отличаются при реализации с использованием очереди?

— Поиск в ширину использует очередь FIFO (First-In-First-Out) для хранения узлов, что гарантирует обработку всех узлов одного уровня перед переходом к следующему.

— Поиск в глубину использует очередь LIFO (Last-In-First-Out) или стек, что позволяет углубляться в одну ветвь, прежде чем переходить к другой.

8. Почему поиск с ограничением глубины не является оптимальным?

Поиск с ограничением глубины может не найти самое короткое решение, так как обрезает все пути, превышающие заданное ограничение, независимо от их потенциальной стоимости.

9. Как итеративное углубление улучшает стандартный поиск с ограничением глубины?

Итеративное углубление постепенно увеличивает предел глубины, начиная с 0, до тех пор, пока решение не будет найдено. Это комбинирует преимущества поиска в глубину и в ширину, избегая проблем с бесконечными ветвями.

10. В каких случаях итеративное углубление становится эффективнее простого поиска в ширину?

Когда память ограничена, так как итеративное углубление использует только пространство для текущего уровня поиска, тогда как поиск в ширину требует хранения всех узлов.

11. Какова основная цель использования алгоритма поиска с ограничением глубины?

Основная цель — эффективно искать решения в графах с большим количеством узлов или с бесконечными ветвями, избегая зацикливания и переполнения памяти.

12. Какие параметры принимает функция depth_limited_search, и каково их назначение?

Функция принимает:

— problem: описание задачи, включая начальное состояние, цель и правила переходов.

— limit: максимальная глубина поиска. Эти параметры используются для определения области поиска и ограничения глубины.

13. Какое значение по умолчанию имеет параметр limit в функции depth_limited_search?

Параметр limit обычно не имеет значения по умолчанию, так как пользователь должен явно задать его. Однако в некоторых реализациях может быть использовано значение, например, 0 или infinity.

14. Что представляет собой переменная frontier, и как она используется в алгоритме?

frontier — это структура данных (например, стек), которая хранит узлы для обработки. Она управляет порядком обработки узлов в процессе поиска.

15. Какую структуру данных представляет LIFOQueue, и почему она используется в этом алгоритме?

LIFOQueue представляет стек (Last-In-First-Out). Она используется, чтобы сначала обрабатывать последний добавленный узел, что соответствует принципу работы поиска в глубину.

16. Каково значение переменной result при инициализации, и что оно означает?

result обычно инициализируется как None, что означает, что результат пока не найден.

17. Какое условие завершает цикл while в алгоритме поиска?

Цикл завершается, если:

— Найден целевой узел.

— Очередь frontier пуста, что означает, что узлы для обработки закончились.

18. Какой узел извлекается с помощью frontier.pop() и почему?

Извлекается последний добавленный узел, так как frontier реализован как стек (LIFO). Это позволяет реализовать углубление в одной ветви.

19. Что происходит, если найден узел, удовлетворяющий условию цели (problem.is_goal(node.state))?

Алгоритм завершает выполнение и возвращает найденный узел как результат.

20. Какую проверку выполняет условие elif len(node) >= limit, и что означает его выполнение?

Это условие проверяет, достиг ли текущий узел предельной глубины поиска. Если да, то дальнейший поиск в этой ветви прекращается.

21. Что произойдет, если текущий узел достигнет ограничения по глубине поиска?

Если узел достиг лимита глубины, он будет обрезан, и его дочерние узлы не будут добавлены в frontier.

22. Какую роль выполняет проверка на циклы elif not is_cycle(node) в алгоритме?

Она предотвращает повторное посещение уже обработанных узлов, избегая зацикливания в графе.

23. Что происходит с дочерними узлами, полученными с помощью функции expand(problem, node)?

Дочерние узлы добавляются в frontier, чтобы они могли быть обработаны на следующих итерациях.

24. Какое значение возвращается функцией, если целевой узел не был найден?

Функция возвращает failure или None, указывая, что решение не найдено.

25. В чем разница между результатами failure и обрезания в алгоритме?

— failure означает, что решение не найдено в графе.

— Обрезание (cutoff) сигнализирует, что поиск был прерван из-за ограничения глубины.

Вывод: в ходе работы были приобретены навыки по работе с поиском с ограничением глубины с помощью языка программирования Python версии 3.x