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 №4
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант

Выполнил:
Степанов Леонид Викторович
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:
Богданов Сергей Сергеевич,
Ассистент департамента цифровых, робототехнических систем и электроники Хацукова А.И

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Исследование поиска с ограничением глубины

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

Порядок выполнения работы:

1. Решите задания лабораторной работы с помощью языка программирования Python и элементов программного кода лабораторной работы 1 (имя файла начинается с PR.AI.001.). Проверьте правильность решения каждой задачи на приведенных тестовых примерах.

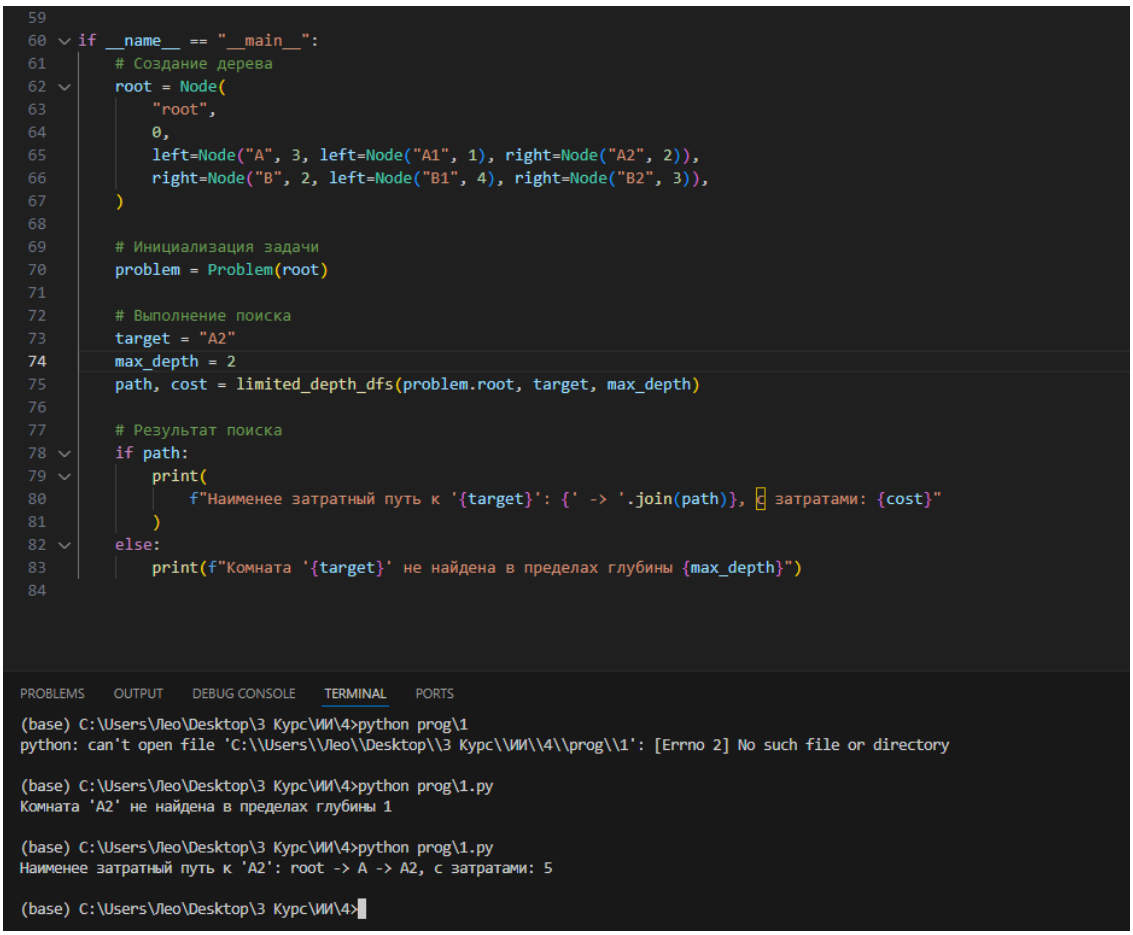
Система навигации робота-пылесоса

```python
class Node:
    def __init__(self, name, cost=0, left=None, right=None):
        self.name = name       # Название узла
        self.cost = cost       # Стоимость узла
        self.left = left       # Левый дочерний узел
        self.right = right     # Правый дочерний узел


class Problem:
    def __init__(self, root):
        self.root = root  # Корневой узел дерева
```

Рисунок 1 – Класс Node и Problem

```python
def limited_depth_dfs(node, target, max_depth, current_depth=0, current_cost=0, path=None):
    if path is None:
        path = []

    # Добавляем текущий узел в путь и суммируем затраты
    path.append(node.name)
    total_cost = current_cost + node.cost

    # Если нашли целевой узел
    if node.name == target:
        return path.copy(), total_cost  # Копируем путь для сохранения результата

    # Если достигли максимальной глубины, возвращаем бесконечную стоимость
    if current_depth >= max_depth:
        path.pop()
        return None, float('inf')

    # Рекурсивный поиск в левом и правом дочернем узле
    left_path, left_cost = (None, float('inf'))
    if node.left:
        left_path, left_cost = limited_depth_dfs(
            node.left, target, max_depth, current_depth + 1, total_cost, path
        )

    right_path, right_cost = (None, float('inf'))
    if node.right:
        right_path, right_cost = limited_depth_dfs(
            node.right, target, max_depth, current_depth + 1, total_cost, path
        )

    # Удаляем текущий узел из пути для корректного возврата
    path.pop()

    # Возвращаем путь с минимальной стоимостью
    if left_cost < right_cost:
        return left_path, left_cost
    else:
        return right_path, right_cost
```

Рисунок 2 – Рекурсивный поиск в глубину с ограничением глубины



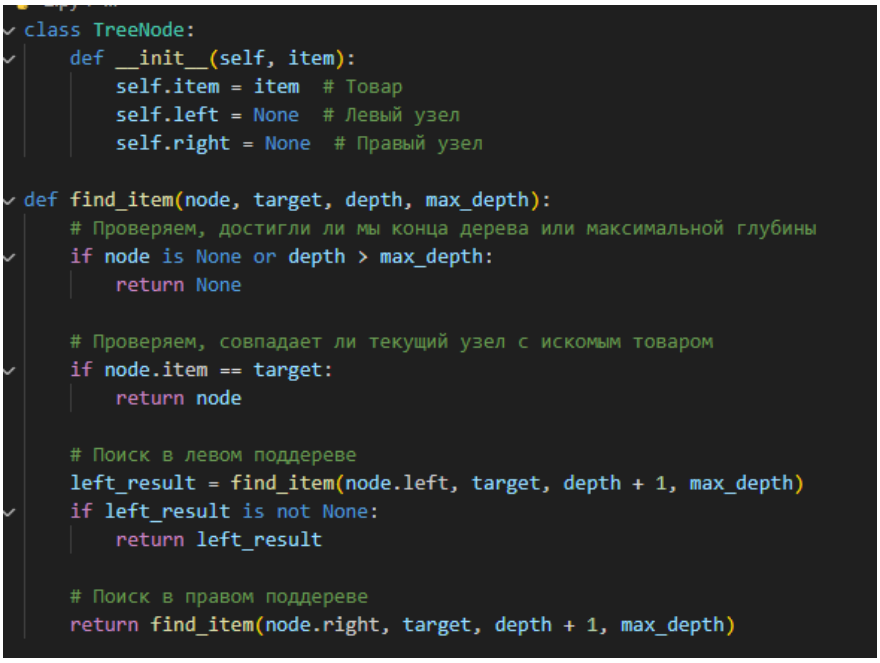
```
59
60 if __name__ == "__main__":
61     # Создание дерева
62     root = Node(
63         "root",
64         0,
65         left=Node("A", 3, left=Node("A1", 1), right=Node("A2", 2)),
66         right=Node("B", 2, left=Node("B1", 4), right=Node("B2", 3)),
67     )
68
69     # Инициализация задачи
70     problem = Problem(root)
71
72     # Выполнение поиска
73     target = "A2"
74     max_depth = 2
75     path, cost = limited_depth_dfs(problem.root, target, max_depth)
76
77     # Результат поиска
78     if path:
79         print(
80             f"Наименее затратный путь к '{target}': {' -> '.join(path)}, с затратами: {cost}"
81         )
82     else:
83         print(f"Комната '{target}' не найдена в пределах глубины {max_depth}")
84
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>python prog\1
python: can't open file 'C:\\Users\\Лео\\Desktop\\3 Курс\\ИИ\\4\\prog\\1': [Errno 2] No such file or directory

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>python prog\1.py
Комната 'A2' не найдена в пределах глубины 1

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>python prog\1.py
Наименее затратный путь к 'A2': root -> A -> A2, с затратами: 5

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>
```

Рисунок 3 – Результат выполнения

На рисунке 3 в первом случае максимальная глубина равна 2 – путь найден, а во втором максимальная глубина 1.

Система управления складом



```
class TreeNode:
    def __init__(self, item):
        self.item = item  # Товар
        self.left = None  # Левый узел
        self.right = None  # Правый узел

def find_item(node, target, depth, max_depth):
    # Проверяем, достигли ли мы конца дерева или максимальной глубины
    if node is None or depth > max_depth:
        return None

    # Проверяем, совпадает ли текущий узел с искомым товаром
    if node.item == target:
        return node

    # Поиск в левом поддереве
    left_result = find_item(node.left, target, depth + 1, max_depth)
    if left_result is not None:
        return left_result

    # Поиск в правом поддереве
    return find_item(node.right, target, depth + 1, max_depth)
```

Рисунок 4 – Склад

Рисунок 5 – Результат работы

Система автоматического управления инвестициями

Представьте, что вы разрабатываете систему для автоматического управления инвестициями, где дерево решений используется для представления последовательности инвестиционных решений и их потенциальных исходов. Цель состоит в том, чтобы найти наилучший исход (максимальную прибыль) на определённой глубине принятия решений, учитывая ограниченные ресурсы и время на анализ.

```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}>"
```

Рисунок 6 – Класс Node

```python
def find_max_at_depth(root, depth):
    if not root:
        return None
    if depth == 0:
        return root.value

    queue = [(root, 0)]
    max_value = float("-inf")

    while queue:
        node, current_depth = queue.pop(0)

        if current_depth == depth:
            max_value = max(max_value, node.value)

        if node.left:
            queue.append((node.left, current_depth + 1))
        if node.right:
            queue.append((node.right, current_depth + 1))

    return max_value if max_value != float("-inf") else None
```

Рисунок 7 – Максимальная грубина

```python
if __name__ == "__main__":
    root = BinaryTreeNode(
        3,
        BinaryTreeNode(1, BinaryTreeNode(0), None),
        BinaryTreeNode(5, BinaryTreeNode(4), BinaryTreeNode(6)),
    )
    limit = 2
    max_value = find_max_at_depth(root, limit)
    print(f"Максимальное значение на указанной глубине: {max_value}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>python prog\1.py
Наименее затратный путь к 'A2': root -> A -> A2, с затратами: 5

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>python prog\3.py
Максимальное значение на указанной глубине: 5

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>python prog\3.py
Максимальное значение на указанной глубине: 6

(base) C:\Users\Лео\Desktop\3 Курс\ИИ\4>
```

Рисунок 8 – Результат работы

2. Для построенного графа лабораторной работы 1 (имя файла начинается с PR.AI.001.) напишите программу на языке программирования Python, которая с помощью алгоритма поиска с ограничением глубины находит минимальное расстояние между начальным и конечным пунктами. Определите глубину дерева поиска, на которой будет найдено решение. Сравните найденное решение с решением, полученным вручную.

```python
# Пример использования
if __name__ == "__main__":
    g = Graph()

    # Добавление рёбер в граф с весами (пример)
    g.add_edge(0, 1, 130)
    g.add_edge(0, 2, 120)
    g.add_edge(1, 3, 90)
    g.add_edge(3, 4, 185)
    g.add_edge(2, 4, 145)
    g.add_edge(4, 5, 300)

    start = 0  # Начальная вершина
    goal = 5   # Конечная вершина
    max_depth = 2  # Максимальная глубина поиска

    found = g.depth_first_search(start, goal, max_depth)

    if found:
        print(f"Путь найден от {start} до {goal} в пределах глубины {max_depth}.")
    else:
        print("Путь не найден в пределах заданной глубины.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

C:\Users\Лео\Desktop\ИИ\4>python .\prog\3.py
Путь найден от 0 до 5 в пределах глубины 3.

C:\Users\Лео\Desktop\ИИ\4>python .\prog\3.py
Путь не найден в пределах заданной глубины.

C:\Users\Лео\Desktop\ИИ\4>
```

Рисунок 9 – Поиск пути с ограничением глубины

Контрольные вопросы:

На изображении представлен код функции `depth_limited_search`, которая выполняет поиск с ограничением глубины. Давайте ответим на вопросы по очереди:

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

Поиск с ограничением глубины — это вариант поиска в глубину, который ограничивает глубину рекурсии (глубину дерева поиска), чтобы избежать бесконечного спуска в дерево, особенно если оно содержит циклы или бесконечные ветви. Ограничение глубины предотвращает зацикливание, обрывая поиск на заданной глубине.

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

Основная цель ограничения глубины — предотвратить зацикливание и сократить время выполнения при работе с бесконечными или очень большими деревьями поиска.

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

Поиск в глубину может бесконечно углубляться в дерево при наличии бесконечных ветвей или циклов. Поиск с ограничением глубины вводит предел глубины и прекращает углубление, когда достигается этот предел.

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

Проверка глубины узла позволяет определить, достигнут ли предел глубины (limit). Если узел на текущей глубине превышает или равен limit, поиск обрезается и возвращает соответствующий статус (обычно cutoff), что предотвращает дальнейшее углубление.

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

Когда достигается лимит глубины, функция возвращает cutoff, что означает, что поиск не может углубляться дальше. Это позволяет обозначить, что поиск не завершился успехом, но и не потерпел неудачу — он прервался из-за ограничения глубины.

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

Если решение находится на глубине, превышающей установленный лимит, то поиск с ограничением глубины не сможет найти это решение, поскольку он остановится до того, как достигнет нужного уровня.

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

Поиск в глубину использует стек или LIFO (last-in, first-out), который обрабатывает последний добавленный узел, позволяя углубляться в дерево.

8. Почему поиск с ограничением глубины не является оптимальным?

Поиск с ограничением глубины не является оптимальным, потому что он может пропустить более короткие пути к цели, если они находятся глубже текущего лимита.

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

Итеративное углубление выполняет поиск с увеличивающимися пределами глубины (1, 2, 3 и так далее), что позволяет достичь любого уровня и найти оптимальное решение, при этом избегая бесконечных циклов. Это дает преимущество поиска в ширину, но с меньшим использованием памяти.

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

Итеративное углубление становится эффективнее, когда дерево поиска очень большое или глубина решения заранее неизвестна. В таких случаях он позволяет контролировать глубину поиска, потребляя меньше памяти, чем поиск в ширину.

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

Цель — поиск цели на фиксированной глубине в дереве или графе, предотвращая зацикливание и сокращая время поиска, особенно в бесконечных или циклических деревьях.

12. Какие параметры принимает функция depth_limited_search и их назначение?

Функция принимает два параметра:

problem — объект задачи, который содержит начальное состояние и целевое состояние.

limit — максимальная глубина поиска (по умолчанию 10), чтобы ограничить глубину дерева.

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

Параметр limit имеет значение по умолчанию 10

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

frontier — это стек (очередь LIFO), который содержит узлы для дальнейшей обработки. В алгоритме frontier используется для хранения узлов и управления порядком их обработки.

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

LIFOQueue представляет собой стек (последний вошел — первый вышел). Он используется для выполнения поиска в глубину, где всегда обрабатывается последний добавленный узел.

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

result инициализируется как failure, что обозначает, что поиск изначально не нашел целевого состояния. Это значение будет возвращено, если поиск не достигнет цели и не выполнит обрезание.

17. Какое условие завершает цикл while в алгоритме поиска?

Цикл while завершается, когда frontier становится пустой, то есть когда больше нет узлов для обработки.

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

frontier.pop() извлекает последний добавленный узел (по принципу LIFO). Это позволяет алгоритму углубляться в дерево, обрабатывая узлы на текущем уровне, прежде чем переходить к соседним узлам.

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

Если найден узел, удовлетворяющий условию цели, он немедленно возвращается, завершая поиск.

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

Это условие проверяет, достиг ли узел установленного предела глубины. Если глубина узла больше или равна limit, поиск прерывается и возвращает cutoff`.

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

Если текущий узел достигает ограничения по глубине, функция отмечает результат как cutoff, прекращает углубление и переходит к обработке других узлов.

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

Проверка elif not is_cycle(node) предотвращает зацикливание, обеспечивая, что узел не будет повторно обработан, если он уже встречался на текущем пути.

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

Дочерние узлы добавляются в frontier для дальнейшей обработки, что позволяет продолжить поиск в глубину по новым ветвям дерева.

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

Если целевой узел не был найден, функция возвращает начальное значение result, которое инициализируется как failure.

25. В чем разница между результатами failure и cutoff в контексте данного алгоритма?

failure указывает, что поиск не нашел целевой узел и достиг конца без успешного завершения.

cutoff обозначает, что поиск был прерван из-за достижения ограничения глубины, но не является окончательной неудачей, так как решение может существовать за пределами указанной глубины.

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