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 №5
дисциплины
«Искусственный интеллект в профессиональной сфере»
Вариант 12.

Выполнила:
Михеева Елена Александровна
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 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 ИТЕРАТИВНЫМ УГЛУБЛЕНИЕМ

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

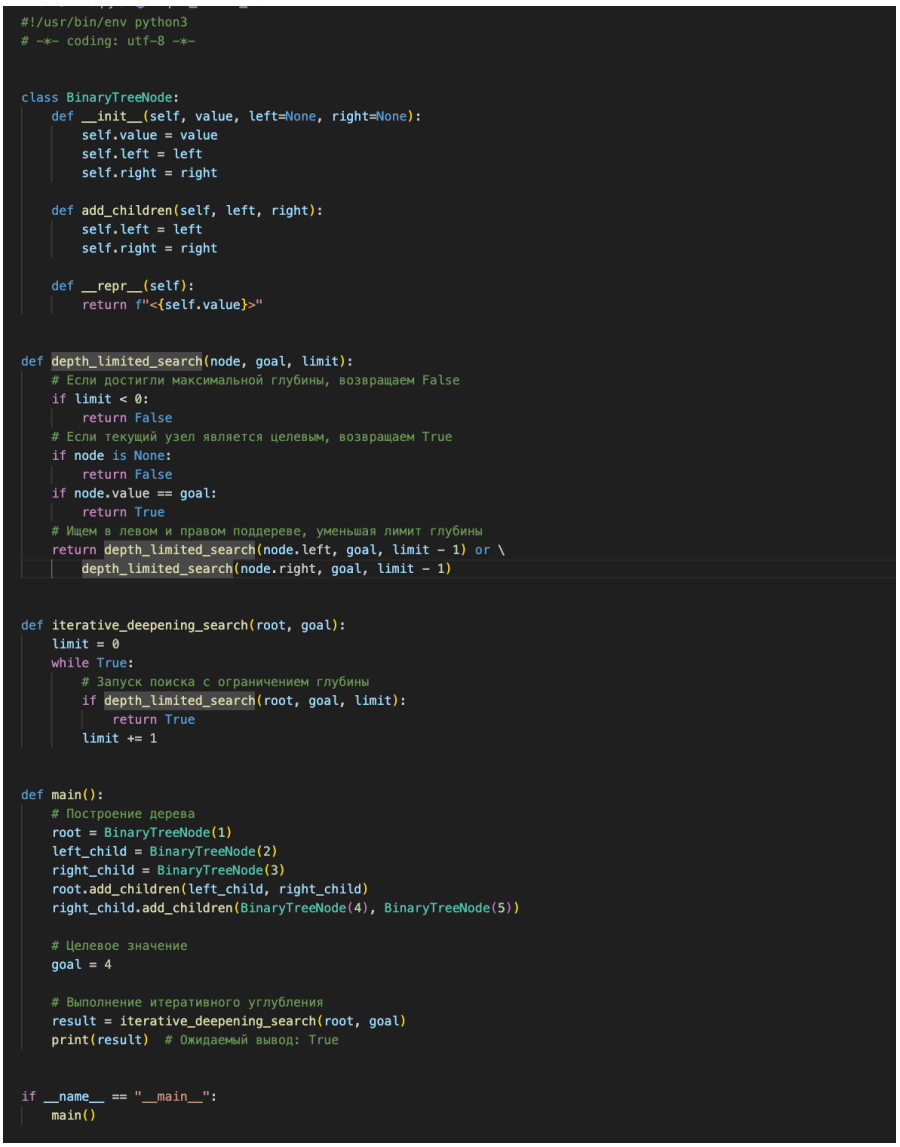
Порядок выполнения работы:

Репозиторий: *https://github.com/helendddd/AI_5.git*

Поиск элемента в дереве с использованием алгоритма итеративного углубления

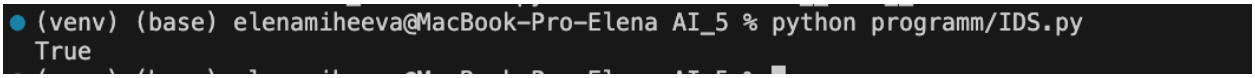
Целью было применение алгоритма итеративного углубления для решения задачи поиска заданного элемента в дереве. Этот метод поиска оказался эффективным для структур данных с неопределенной или большой глубиной, позволяя находить элементы при минимальном времени поиска и объемах памяти.

В задаче была рассмотрена система управления доступом, где каждый пользователь был представлен узлом в дереве, и каждый узел содержал уникальный идентификатор пользователя. Необходимо было разработать метод поиска, который позволял проверить существование пользователя с заданным идентификатором в системе, используя структуру дерева и алгоритм итеративного углубления.



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


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

    def add_children(self, left, right):
        self.left = left
        self.right = right

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


def depth_limited_search(node, goal, limit):
    # Если достигли максимальной глубины, возвращаем False
    if limit < 0:
        return False
    # Если текущий узел является целевым, возвращаем True
    if node is None:
        return False
    if node.value == goal:
        return True
    # Ищем в левом и правом поддереве, уменьшая лимит глубины
    return depth_limited_search(node.left, goal, limit - 1) or \
        depth_limited_search(node.right, goal, limit - 1)


def iterative_deepening_search(root, goal):
    limit = 0
    while True:
        # Запуск поиска с ограничением глубины
        if depth_limited_search(root, goal, limit):
            return True
        limit += 1


def main():
    # Построение дерева
    root = BinaryTreeNode(1)
    left_child = BinaryTreeNode(2)
    right_child = BinaryTreeNode(3)
    root.add_children(left_child, right_child)
    right_child.add_children(BinaryTreeNode(4), BinaryTreeNode(5))

    # Целевое значение
    goal = 4

    # Выполнение итеративного углубления
    result = iterative_deepening_search(root, goal)
    print(result)  # Ожидаемый вывод: True


if __name__ == "__main__":
    main()
```

Рисунок 1. Код программы



```
(venv) (base) elenamiheeva@MacBook-Pro-Elena AI_5 % python programm/IDS.py
True
```

Рисунок 2. Результат работы программы

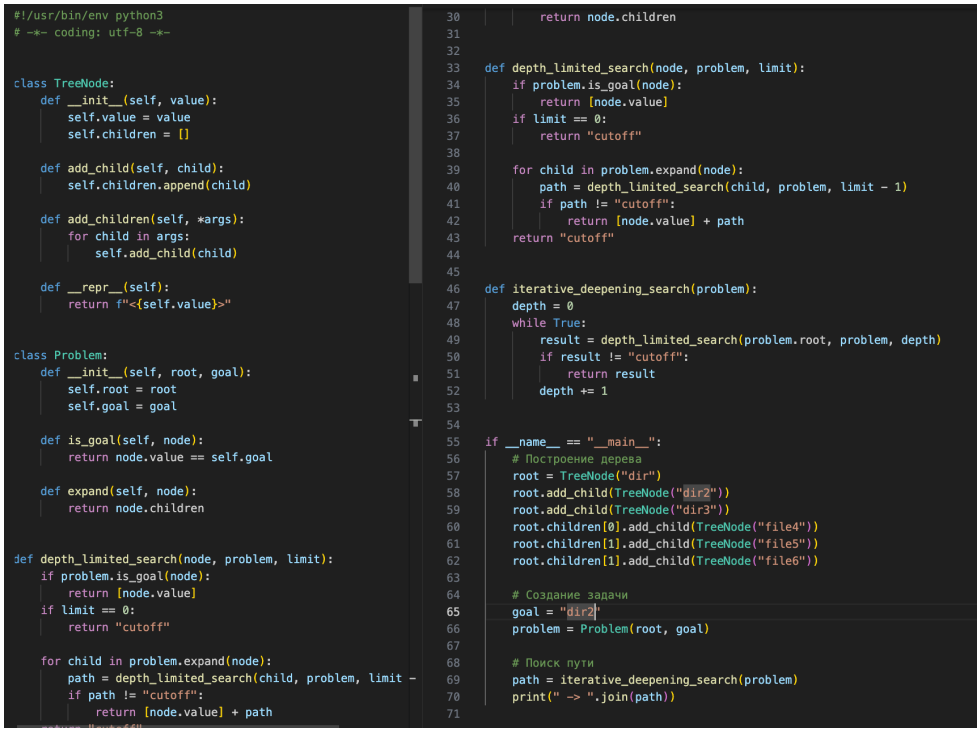
Поиск в файловой системе.

Целью было ознакомление с применением алгоритма итеративного углубления для нахождения пути от корневого узла до целевого узла в дереве. Этот метод продемонстрировал свою эффективность в ситуациях с неопределенной глубиной структуры данных, обеспечивая поиск с минимальными затратами памяти.

Рассматривалась задача поиска информации в иерархических структурах данных, например, в файловой системе, где каждый каталог может

содержать подкаталоги и файлы. Алгоритм итеративного углубления оказался идеальным для таких задач, поскольку он позволил исследовать структуру данных постепенно, углубляясь на один уровень за раз и возвращаясь, если целевой узел не был найден.

Для решения задачи было построено дерево, где каждый узел представлял каталог в файловой системе, а целью поиска стал определенный файл. Для поиска пути от корневого каталога до каталога или файла, содержащего искомый файл, был использован алгоритм итеративного углубления.



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []

    def add_child(self, child):
        self.children.append(child)

    def add_children(self, *args):
        for child in args:
            self.add_child(child)

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


class Problem:
    def __init__(self, root, goal):
        self.root = root
        self.goal = goal

    def is_goal(self, node):
        return node.value == self.goal

    def expand(self, node):
        return node.children


def depth_limited_search(node, problem, limit):
    if problem.is_goal(node):
        return [node.value]
    if limit == 0:
        return "cutoff"

    for child in problem.expand(node):
        path = depth_limited_search(child, problem, limit -
        if path != "cutoff":
            return [node.value] + path
```

```python
        return node.children


def depth_limited_search(node, problem, limit):
    if problem.is_goal(node):
        return [node.value]
    if limit == 0:
        return "cutoff"

    for child in problem.expand(node):
        path = depth_limited_search(child, problem, limit - 1)
        if path != "cutoff":
            return [node.value] + path
    return "cutoff"


def iterative_deepening_search(problem):
    depth = 0
    while True:
        result = depth_limited_search(problem.root, problem, depth)
        if result != "cutoff":
            return result
        depth += 1


if __name__ == "__main__":
    # Построение дерева
    root = TreeNode("dir")
    root.add_child(TreeNode("dir2"))
    root.add_child(TreeNode("dir3"))
    root.children[0].add_child(TreeNode("file4"))
    root.children[1].add_child(TreeNode("file5"))
    root.children[1].add_child(TreeNode("file6"))

    # Создание задачи
    goal = "dir2"
    problem = Problem(root, goal)

    # Поиск пути
    path = iterative_deepening_search(problem)
    print(" -> ".join(path))
```

Рисунок 3. Код программы

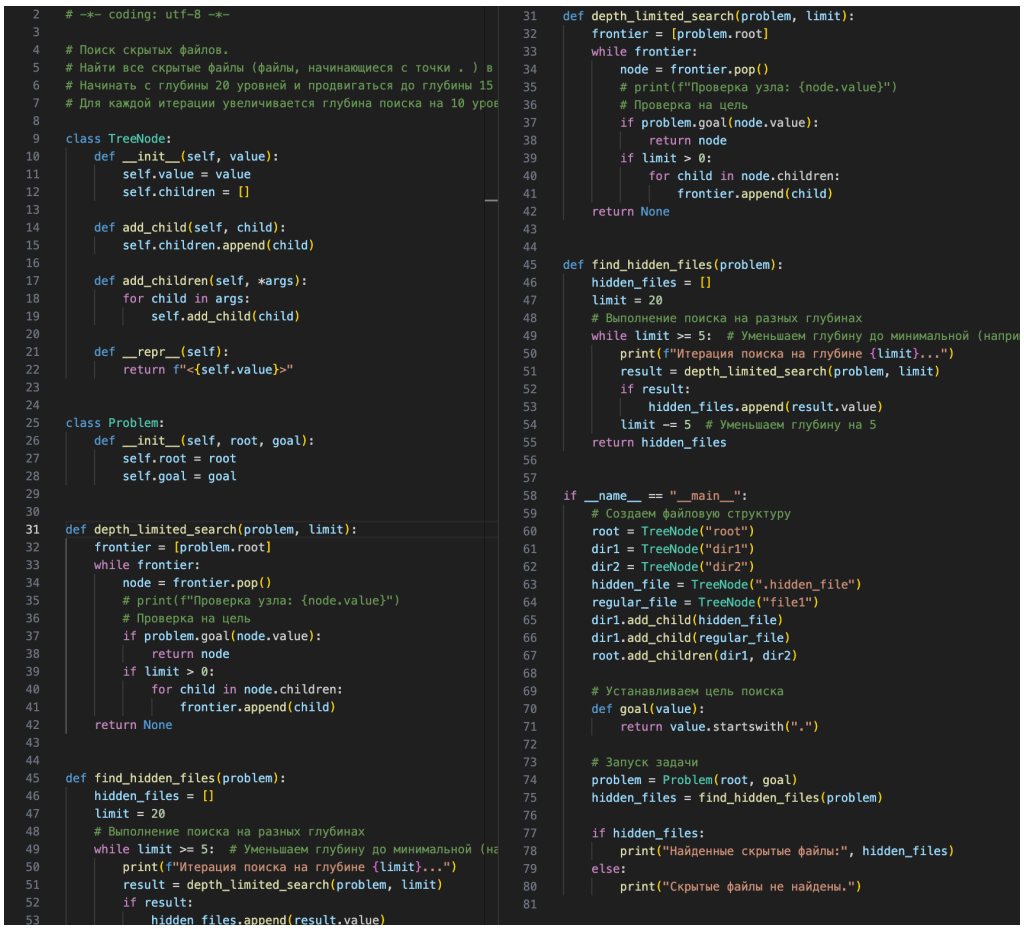
```
(venv) (base) elenamiheeva@MacBook-Pro-Elena AI_5 % python programm/file_search.py
dir -> dir3 -> file5
(venv) (base) elenamiheeva@MacBook-Pro-Elena AI_5 % python programm/file_search.py
dir -> dir2
```

Рисунок 4. Результат работы программы

Индивидуальное задание.

Поиск скрытых файлов.

Найти все скрытые файлы (файлы, начинающиеся с точки) в файловом дереве. Начало с глубины 20 уровней и продвижение до глубины 15 уровней. Для каждой итерации увеличивалась глубина поиска на 10 уровней.



```python
# -*- coding: utf-8 -*-

# Поиск скрытых файлов.
# Найти все скрытые файлы (файлы, начинающиеся с точки . ) в
# Начинать с глубины 20 уровней и продвигаться до глубины 15
# Для каждой итерации увеличивается глубина поиска на 10 уров

class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []

    def add_child(self, child):
        self.children.append(child)

    def add_children(self, *args):
        for child in args:
            self.add_child(child)

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


class Problem:
    def __init__(self, root, goal):
        self.root = root
        self.goal = goal


def depth_limited_search(problem, limit):
    frontier = [problem.root]
    while frontier:
        node = frontier.pop()
        # print(f"Проверка узла: {node.value}")
        # Проверка на цель
        if problem.goal(node.value):
            return node
        if limit > 0:
            for child in node.children:
                frontier.append(child)
    return None


def find_hidden_files(problem):
    hidden_files = []
    limit = 20
    # Выполнение поиска на разных глубинах
    while limit >= 5:  # Уменьшаем глубину до минимальной (на
        print(f"Итерация поиска на глубине {limit}...")
        result = depth_limited_search(problem, limit)
        if result:
            hidden_files.append(result.value)
        limit -= 5  # Уменьшаем глубину на 5
    return hidden_files


if __name__ == "__main__":
    # Создаем файловую структуру
    root = TreeNode("root")
    dir1 = TreeNode("dir1")
    dir2 = TreeNode("dir2")
    hidden_file = TreeNode(".hidden_file")
    regular_file = TreeNode("file1")
    dir1.add_child(hidden_file)
    dir1.add_child(regular_file)
    root.add_children(dir1, dir2)

    # Устанавливаем цель поиска
    def goal(value):
        return value.startswith(".")

    # Запуск задачи
    problem = Problem(root, goal)
    hidden_files = find_hidden_files(problem)

    if hidden_files:
        print("Найденные скрытые файлы:", hidden_files)
    else:
        print("Скрытые файлы не найдены.")
```

Рисунок 5. Код программы для задания

```
(venv) (base) elenamiheeva@MacBook-Pro-Elena AI_5 % python programm/hidden_file.py
Найденные скрытые файлы: ['.hidden_file', '.hidden_file', '.hidden_file', '.hidden_file']
```

Рисунок 6. Результат работы программы

Ответы на контрольные вопросы:

1. Что означает параметр n в контексте поиска с ограниченной глубиной, и как он влияет на поиск?

Параметр n — это максимальная глубина, до которой будет выполняться поиск. Он ограничивает количество уровней, на которых будут проверяться узлы, предотвращая бесконечное расширение дерева.

2. Почему невозможно заранее установить оптимальное значение для глубины d в большинстве случаев поиска?

Оптимальное значение глубины неизвестно заранее, так как оно зависит от структуры данных и местоположения решения, что невозможно точно предсказать.

3. Какие преимущества дает использование алгоритма итеративного углубления по сравнению с поиском в ширину?

Итеративное углубление требует меньше памяти, поскольку сохраняет только текущий уровень дерева, в отличие от поиска в ширину, который требует хранения всех узлов на каждом уровне.

4. Опишите, как работает итеративное углубление и как оно помогает избежать проблем с памятью.

Алгоритм итеративного углубления выполняет несколько проходов поиска с ограничением глубины. Каждый проход ищет решение на новой, увеличенной глубине, минимизируя использование памяти, так как сохраняет только текущий уровень.

5. Почему алгоритм итеративного углубления нельзя просто продолжить с текущей глубины, а приходится начинать поиск заново с корневого узла?

Каждый новый проход с увеличенной глубиной не может использовать информацию о предыдущих уровнях, так как структура поиска может изменяться, и требуется новая проверка для всех узлов.

6. Какие временные и пространственные сложности имеет поиск с итеративным углублением?

Время работы — O(b^d), где b — коэффициент разветвления, а d — глубина решения. Пространственная сложность — O(b*d), так как сохраняются узлы на каждом уровне.

7. Как алгоритм итеративного углубления сочетает в себе преимущества поиска в глубину и поиска в ширину?

Итеративное углубление сочетает низкое потребление памяти (поиск в глубину) с гарантией нахождения решения на минимальной глубине (поиск в ширину).

8. Почему поиск с итеративным углублением остается эффективным, несмотря на повторное генерирование дерева на каждом шаге увеличения глубины?

Хотя дерево генерируется заново, итерации поиска с ограничением глубины обычно не требуют значительных затрат по времени, поскольку глубина постепенно увеличивается.

9. Как коэффициент разветвления b и глубина d влияют на общее количество узлов, генерируемых алгоритмом итеративного углубления?

Общее количество узлов растет экспоненциально с увеличением коэффициента разветвления b и глубины d, поскольку на каждом уровне дерева появляется b новых узлов.

10. В каких ситуациях использование поиска с итеративным углублением может быть не оптимальным, несмотря на его преимущества?

Если решение находится на большой глубине, итеративное углубление может быть медленным, так как много раз будет выполняться повторный поиск на маленьких глубинах.

11. Какую задачу решает функция iterative_deepening_search?

Функция выполняет итеративное углубление, постепенно увеличивая глубину поиска, чтобы найти решение задачи, начиная с минимальной глубины.

12. Каков основной принцип работы поиска с итеративным углублением?

Алгоритм выполняет несколько поисков с ограничением глубины, начиная с глубины 1, затем 2 и так далее, пока не будет найдено решение.

13. Что представляет собой аргумент problem, передаваемый в функцию iterative_deepening_search?

Аргумент problem представляет собой задачу, которая содержит начальное состояние и цель поиска, а также методы для проверки состояния и расширения узлов.

14. Какова роль переменной limit в алгоритме?

Переменная limit определяет максимальную глубину поиска в текущей итерации алгоритма.

15. Что означает использование диапазона range(1, sys.maxsize) в цикле for?

Этот диапазон задает последовательность глубин для поиска, начиная с 1 и до максимально возможной глубины.

16. Почему предел глубины поиска увеличивается постепенно, а не устанавливается сразу на максимальное значение?

Постепенное увеличение глубины позволяет более эффективно исследовать дерево, начиная с более поверхностных узлов, что улучшает использование памяти и времени.

17. Какая функция вызывается внутри цикла и какую задачу она решает?

Внутри цикла вызывается функция depth_limited_search, которая выполняет поиск с ограничением глубины, проверяя узлы до указанного предела.

18. Что делает функция depth_limited_search, и какие результаты она может возвращать?

Функция выполняет поиск узлов на глубину, ограниченную параметром limit, и возвращает найденный узел или None, если решение не найдено.

19. Какое значение представляет собой cutoff, и что оно обозначает в данном алгоритме?

Значение cutoff обозначает случай, когда поиск не достиг заданной глубины, и решение не найдено на текущем уровне.

20. Почему результат сравнивается с cutoff перед тем, как вернуть результат?

Это необходимо для корректного определения, было ли найдено решение или поиск достиг лимита глубины, что сигнализирует о необходимости продолжить поиск на более глубоких уровнях.

21. Что произойдет, если функция depth_limited_search найдет решение на первой итерации?

Если решение найдено, функция сразу возвращает результат, и дальнейшие итерации поиска не выполняются.

22. Почему функция может продолжать выполнение до тех пор, пока не достигнет sys.maxsize?

Это позволяет выполнить поиск на всех возможных уровнях дерева до максимально возможной глубины, гарантируя нахождение решения.

23. Каковы преимущества использования поиска с итеративным углублением по сравнению с обычным поиском в глубину?

Поиск с итеративным углублением экономит память, так как не требует хранения всех узлов на каждом уровне, как в поиске в ширину.

24. Какие потенциальные недостатки может иметь этот подход?

Повторное выполнение поиска на каждом уровне может быть менее эффективным, если решение находится на глубокой, но не слишком глубокой уровне.

25. Как можно оптимизировать данный алгоритм для ситуаций, когда решение находится на больших глубинах?

Для оптимизации можно использовать другие методы поиска, такие как двоичный поиск или эвристические методы, чтобы сократить количество проверяемых уровней.

Вывод: были приобретены навыки по работе с поиском с интерактивным углублением с помощью языка программирования Python версии 3.x