

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

тут перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**  
**дисциплины «Искусственный интеллект в профессиональной сфере»**  
**Вариант 2**

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:  
Воронкин Роман Александрович,  
доцент департамента цифровых,  
робототехнических систем и  
электроники

---

(подпись)

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

Ставрополь, 2024 г.

**Тема:** «Исследование поиска с ограничением глубины»

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

Порядок выполнения работы:

Задача 1. Поиск элемента в дереве с использованием алгоритма итеративного углубления. Необходимо разработать метод поиска для системы управления доступом, где каждый пользователь представлен узлом в дереве, а каждый узел содержит уникальный идентификатор пользователя. Задача заключается в том, чтобы проверить наличие пользователя с указанным идентификатором в системе, используя структуру дерева и алгоритм итеративного углубления.

Листинг 1 – Код программы `general_tree.py`

```
#!/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 iterative_deepening_search(root, goal):
    def depth_limited_search(node, goal, limit):
        if node is None:
            return False
        if node.value == goal:
            return True
        if limit == 0:
            return False
        return (
            depth_limited_search(node.left, goal, limit - 1) or
            depth_limited_search(node.right, goal, limit - 1)
        )
```

```

depth = 0
while True:
    found = depth_limited_search(root, goal, depth)
    if found:
        return True
    depth += 1

# 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

# Поиск с итеративным углублением
result = iterative_deepening_search(root, goal)

print(result)

```

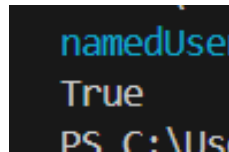


Рисунок 1 – Вывод программы

Задача 2. Поиск в файловой системе. Рассмотрим задачу поиска информации в иерархических структурах данных, например, в файловой системе, где каждый каталог может содержать подкаталоги и файлы. Алгоритм итеративного углубления идеально подходит для таких задач, поскольку он позволяет исследовать структуру данных постепенно, углубляясь на один уровень за раз и возвращаясь, если целевой узел не найден. Для этого необходимо:

- Построить дерево, где каждый узел представляет каталог в файловой системе, а цель поиска — определенный файл.
- Найти путь от корневого каталога до каталога (или файла), содержащего искомый файл, используя алгоритм итеративного углубления.

## Листинг 2 – Код программы file\_system.py

```
#!/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}>"

def iterative_deepening_search(root, goal):
    def depth_limited_search(node, goal, limit, path):
        if node is None:
            return False
        path.append(node.value)
        if node.value == goal:
            return True
        if limit == 0:
            path.pop()
            return False
        for child in node.children:
            if depth_limited_search(child, goal, limit - 1, path):
                return True
        path.pop()
        return False

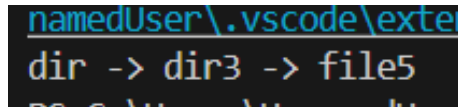
    depth = 0
    while True:
        path = []
        found = depth_limited_search(root, goal, depth, path)
        if found:
            return " -> ".join(path)
        depth += 1

# 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"))
```

```
# Цель поиска
goal = "file5"

# Поиск с итеративным углублением
result = iterative_deepening_search(root, goal)

print(result)
```



```
namedUser\.vscode\exten
dir -> dir3 -> file5
PS C:\Users\namedUser>
```

Рисунок 2 – Результат вывода программы

Индивидуальное задание. Поиск файлов с дублирующимся содержимым. Найдите два разных файла в файловом дереве, которые имеют одинаковое содержимое (по побайтовому сравнению), используя итеративное углубление. Дерево более ста файлов, и их глубина может достигать 10 уровней.

Листинг 3 – Код программы individual.py

```
import os
import hashlib

def find_duplicate_files(directory):
    file_hashes = {}
    # Итеративное углубление с использованием стека
    stack = [directory]

    while stack:
        current_dir = stack.pop()
        try:
            for entry in os.scandir(current_dir):
                if entry.is_dir():
                    stack.append(entry.path)
                elif entry.is_file():
                    file_hash = hash_file(entry.path)
                    if file_hash in file_hashes:
                        print(f"Найдены дублирующиеся файлы: {entry.path} и {file_hashes[file_hash]}")
                        return
                    file_hashes[file_hash] = entry.path
            except PermissionError:
                # Пропускаем файлы и директории, доступ к которым ограничен
                continue
```

```
def hash_file(file_path):
    """Вычисление хеша файла для сравнения содержимого"""
    hash_sha256 = hashlib.sha256()
    with open(file_path, 'rb') as file:
        for chunk in iter(lambda: file.read(4096), b''):
            hash_sha256.update(chunk)
    return hash_sha256.hexdigest()

if __name__ == "__main__":
    directory = '<путь до директории>'
    find_duplicate_files(directory)
```

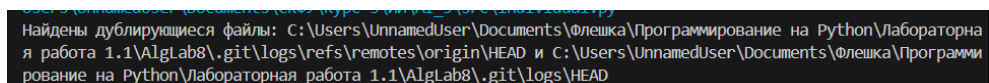


Рисунок 3 – Вывод программы

### Ответы на контрольные вопросы:

Ответ на вопросы:

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

Параметр  $n$  в контексте поиска с ограниченной глубиной обычно означает максимально допустимую глубину для данного поиска. Этот параметр ограничивает количество уровней, которые могут быть исследованы в процессе поиска. Чем меньше значение  $n$ , тем более ограниченным будет поиск, а чем оно больше, тем больше уровней будет охвачено.

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

Невозможно заранее установить оптимальное значение для глубины  $d$ , потому что в большинстве задач поиска глубина решения может быть неизвестна или зависеть от множества факторов, таких как структура пространства поиска или специфика проблемы. Кроме того, в некоторых случаях глубина решения может быть сильно переменной и даже неопределённой.

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

Алгоритм итеративного углубления имеет несколько преимуществ по сравнению с поиском в ширину. Во-первых, он использует гораздо меньше памяти, поскольку в каждый момент времени в памяти хранится только один уровень дерева поиска. Во-вторых, его время выполнения в худшем случае аналогично времени работы поиска в ширину, при этом итерации осуществляются по очереди, начиная с меньших глубин.

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

Итеративное углубление работает, выполняя последовательные поиски в глубину с увеличивающейся максимальной глубиной. Это позволяет избежать переполнения памяти, так как каждый поиск ограничен глубиной, а в памяти хранится только текущий уровень дерева поиска. Вместо того чтобы строить всё дерево, как в поиске в ширину, итеративное углубление эффективно управляет использованием памяти.

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

Алгоритм итеративного углубления не может продолжить поиск с текущей глубины, потому что поиск в глубину выполняется в одном направлении до достижения указанной глубины. Для того чтобы исследовать более глубокие уровни, нужно начать новый поиск с корня, чтобы не забыть проверенные на более мелких уровнях.

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

Временная сложность поиска с итеративным углублением — это сумма временных сложностей для всех уровней поиска. В худшем случае она составляет  $O(b^d)$   $b$  — это коэффициент разветвления, а  $d$  — максимальная глубина. Пространственная сложность составляет  $O(b \cdot d)$ , так как в памяти хранится только текущий уровень.

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

Алгоритм итеративного углубления сочетает преимущества поиска в глубину (меньшее потребление памяти) с преимуществами поиска в ширину (обеспечивает нахождение оптимального решения при равных затратах).

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

Поиск с итеративным углублением остаётся эффективным, несмотря на повторное генерирование дерева, потому что на каждом шаге увеличивается только глубина поиска, а не количество возможных путей. Алгоритм оптимизирует память за счёт того, что каждый поиск ограничен только текущей глубиной, и не хранит всю структуру дерева поиска.

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

Коэффициент разветвления  $b$  и глубина  $d$  определяют количество узлов, генерируемых алгоритмом. Количество узлов будет пропорционально  $O(b^d)$ , что означает экспоненциальный рост числа узлов с увеличением глубины и коэффициента разветвления.

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



Алгоритм итеративного углубления может быть не оптимальным в случае, если решение находится на большой глубине, а простое увеличение глубины на каждом шаге приводит к излишним вычислениям. В таких случаях более эффективными могут быть другие алгоритмы, такие как поиск с приоритетами или алгоритм A\*.

11. Какую задачу решает функция `iterative_deepening_search`?

Функция `iterative_deepening_search` решает задачу поиска решения с помощью итеративного углубления, постепенно увеличивая предел глубины до тех пор, пока не будет найдено решение.

12. Каков основной принцип работы поиска с итеративным углублением?

Основной принцип работы поиска с итеративным углублением заключается в выполнении повторяющихся поисков в глубину с постепенно увеличиваемым пределом глубины до тех пор, пока не будет найдено решение или не будет достигнут предел глубины.

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

Аргумент `problem`, передаваемый в функцию `iterative_deepening_search`, представляет собой описание задачи или проблемы, для которой требуется найти решение. Это может включать начальное состояние, цель и методы перехода.

14. Какова роль переменной `limit` в алгоритме?

Переменная `limit` в алгоритме задаёт максимальную глубину, до которой будет идти поиск на каждом шаге итеративного углубления.

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

Диапазон `range(1, sys.maxsize)` в цикле используется для поочередного увеличения предела глубины в поиске. Это позволяет постепенно увеличивать глубину поиска на каждом шаге.

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

Предел глубины поиска увеличивается постепенно, а не устанавливается сразу на максимальное значение, потому что это позволяет избежать чрезмерных вычислительных затрат. Постепенное увеличение позволяет сначала найти решение на более мелких уровнях и только потом увеличивать глубину, если решение не найдено.

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

Внутри цикла вызывается функция поиска в глубину с ограниченной глубиной (например, `depth_limited_search`). Эта функция выполняет поиск до указанной глубины и возвращает результат.

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

Функция `depth_limited_search` ограничивает поиск заданной глубиной и может возвращать либо решение, либо сигнал о том, что решение не найдено на текущем уровне.

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

Значение `cutoff` обозначает предельное значение глубины, до которой выполняется поиск. Если глубина превышает `cutoff`, поиск не продолжается.

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

Результат сравнивается с `cutoff` перед возвратом результата для проверки того, не превышает ли текущая глубина допустимого лимита.

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

Если функция `depth_limited_search` находит решение на первой итерации, то оно немедленно возвращается как результат поиска, без дальнейших шагов.

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

Функция может продолжать выполнение до тех пор, пока не достигнет `sys.maxsize`, так как это максимальный предел для глубины в рамках текущей реализации, и позволяет алгоритму продолжать поиск до тех пор, пока не будет найдена цель или не достигнут предел.

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

Преимущества использования поиска с итеративным углублением по сравнению с обычным поиском в глубину заключаются в меньших требованиях к памяти и способности эффективно искать решение на различных уровнях.

24. Какие потенциальные недостатки может иметь этот подход?

Потенциальные недостатки включают повторное генерирование дерева на каждом шаге, что может привести к большому количеству избыточных вычислений при глубоком поиске.

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

Чтобы оптимизировать алгоритм для ситуаций, когда решение находится на больших глубинах, можно комбинировать его с методами, такими как использование эвристических функций или ограничение пространства поиска с помощью других техник, таких как A\*.

**Выводы:** В процессе выполнения лабораторной работы были приобретены навыки по работе с поиском с итеративным углублением с помощью языка программирования Python версии 3.x, проработаны примеры и выполнены индивидуальные задания.

Ссылка на репозиторий:

[I\UnnamedUserI\AI\\_5: Искусственный интеллект в профессиональной сфере.](#)  
[Лабораторная работа №5](#)