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
По дисциплине
«Искусственный интеллект в профессиональной сфере»

Выполнил:
Арзютов Иван Владиславович

3 курс, группа ИТС-б-о-22-1,

11.03.02 «Инфокоммуникационные технологии и системы связи, очная форма обучения

_____
(подпись)

Проверил:

Воронкин Р.А., канд. тех. наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Исследование поиска с итеративным углублением

Ссылка на репозиторий: https://github.com/Ivanuschka/5-LB

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

Порядок выполнения работы

Задание 1

Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT.

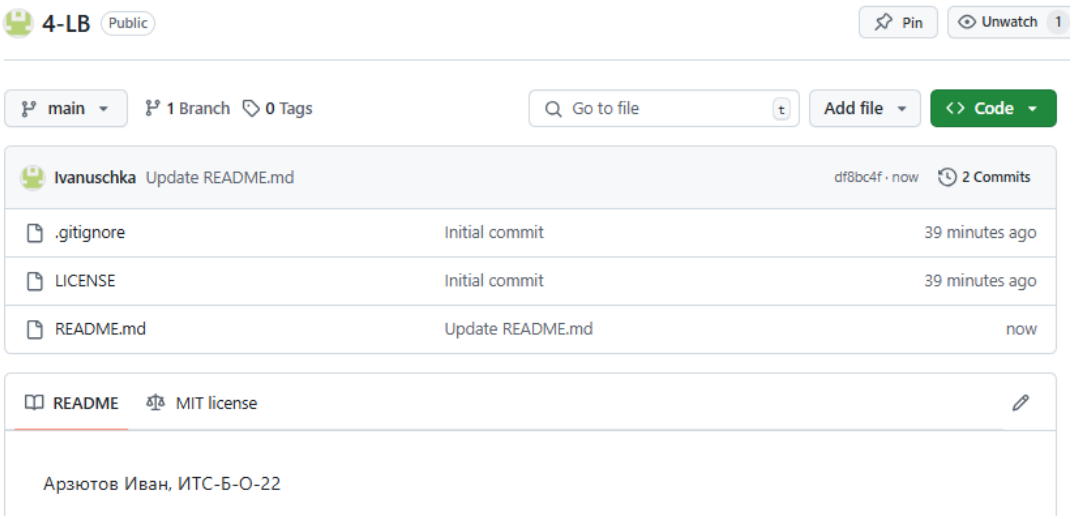

Рисунок 1. Создан новый репозиторий

Задание 2

Клонирование репозитория на свой компьютер.

```
C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа>git clone https://github.com/Ivanuschka/5-LB.git
Cloning into '5-LB'...
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)
Receiving objects: 100% (5/5), done.

C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа>cd C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная
работа\5-LB

C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа\5-LB>
```

Рисунок 2. Клонирование репозитория

Задание 3

Проработка примера лабораторной работы

```python
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}>"
# 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
```

Рисунок 3. Выполнение примера – Поиск элемента древа с использованием алгоритма итеративного углубления

Задание 5

```python
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}>"
# 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"
```

Рисунок 4. Выполнение примера – Поиск в файловой системе

Задание 6. Индивидуальное

Поиск файла с определённым расширением. В файловом дереве существует множество файлов с разными расширениями. Найти все файлы с расширением .log , используя алгоритм итеративного углубления. Ограничение: дерево содержит не менее 10 уровней.

```python
import os
import random
import string

# Глобальная переменная для хранения найденных файлов
found_files = []

def create_random_directory_tree(base_path, depth=10, max_subdirs=3, max_files=5):
    """Рекурсивно создаёт случайную структуру каталогов и файлов"""
    if depth == 0:
        return

    os.makedirs(base_path, exist_ok=True)

    # Генерируем случайные файлы
    extensions = ['.txt', '.log', '.csv', '.json', '.xml']
    for _ in range(random.randint(1, max_files)):
        filename = ''.join(random.choices(string.ascii_letters, k=8)) + random.choice(extensions)
        with open(os.path.join(base_path, filename), 'w') as f:
            f.write("Sample data")

    # Генерируем случайные подкаталоги
    for _ in range(random.randint(1, max_subdirs)):
        subdir_name = ''.join(random.choices(string.ascii_letters, k=5))
        create_random_directory_tree(os.path.join(base_path, subdir_name), depth - 1, max_subdirs, max_files)


def iterative_deepening_search(root_path, extension=".log", max_depth=10):
    """Поиск файлов с заданным расширением методом итеративного углубления"""
    for depth in range(1, max_depth + 1):
        found = depth_limited_search(root_path, extension, depth)
        if found:
            return found
    return None


def depth_limited_search(path, extension, depth):
    """Поиск с ограничением глубины"""
    global found_files
    if depth < 0:
        return False

    try:
        for entry in os.scandir(path):
            if entry.is_file() and entry.name.endswith(extension):
                found_files.append(entry.path)
            elif entry.is_dir():
                depth_limited_search(entry.path, extension, depth - 1)
    except PermissionError:
        pass

    return bool(found_files)


# Создаём виртуальное дерево файлов в папке "virtual_disk"
base_directory = "virtual_disk"
create_random_directory_tree(base_directory)

# Запускаем поиск файлов с расширением .log
iterative_deepening_search(base_directory, ".log")

# Вывод результатов
print("\nНайденные файлы .log:")
for file in found_files:
    print(file)
```

Рисунок 5. Поиск по дереву каталогов с итеративным углублением

Вывод по лабораторной работе: в ходе выполнения лабораторной работы были приобретены навыки по работе с поиском с итеративным углублением с помощью языка программирования Python версии 3.x

Ответы на контрольные вопросы

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

Параметр n представляет собой глубину, до которой осуществляется поиск. Он влияет на объем дерева, которое будет исследовано, и может ограничить количество узлов, которые будут проверены. Более высокие значения n позволят исследовать более глубокие уровни дерева, но могут также увеличивать затраты по времени и памяти.

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

Оптимальное значение глубины d часто неизвестно заранее, так как в различных задачах решения могут находиться на разных уровнях глубины. Кроме того, если структура дерева не известна (например, может быть очень глубокой или широкой), трудно предсказать, где будет находиться решение.

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

Итеративное углубление требует значительно меньше памяти, так как хранит в памяти только узлы текущего уровня поиска, как и поиск в глубину. В отличие от поиска в ширину, оно не требует большого объема памяти для хранения всех узлов на текущем уровне, что делает его более подходящим для глубоких деревьев.

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

Итеративное углубление последовательно выполняет поиск в глубину с увеличивающейся глубиной. На каждой итерации начинается новый поиск с корневого узла до заданной глубины. Это позволяет избежать хранения всех узлов в памяти, так как на каждом уровне рассматриваются лишь узлы до текущей глубины.

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

Поскольку при каждом увеличении глубины необходимо пересоздать дерево для нового уровня поиска. Структура решений может

отличаться на разных уровнях, и алгоритму нужно гарантия, что все узлы на текущем уровне будут проверены перед углублением.

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

Временная сложность поиска с итеративным углублением составляет O(b^d), где b — коэффициент разветвления, а d — глубина. Пространственная сложность составляет O(b*d), что значительно меньше, чем у поиска в ширину.

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

Он использует метод глубинного поиска для минимизации потребления памяти, но одновременно обеспечивает возможность находить решения, находящиеся на более мелких уровнях, как в случае поиска в ширину.

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

Повторное генерирование дерева снижается по сравнению с полным исследованием всех узлов, так как многие узлы могут быть пройдены и не требуются для дальнейшего поиска, что делает его достаточно эффективным на практике.

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

Общее количество узлов, генерируемых алгоритмом, может быть охарактеризовано как сумма всех узлов на каждом уровне от 0 до d. Это означает, что количество узлов будет расти экспоненциально в зависимости от b и d, что влияет на время выполнения.

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

Если решение находится глубоко в дереве, и b (коэффициент разветвления) велик, то временные затраты на повторное генерирование узлов могут оказаться значительными. Также, если мы точно знаем, что глубина решения невелика, то более простой подход поиска в глубину может быть эффективнее.

11. Какую задачу решает функция iterative_deepening_search?

Функция iterative_deepening_search решает задачу поиска решения в графе или дереве, используя метод итеративного углубления, последовательно увеличивая глубину и проверяя узлы.

12. Каков основной принцип работы поиска с итеративным углублением?

Основной принцип заключается в том, чтобы исполнять серию поисков в глубину с постепенно увеличивающейся глубиной, начиная с нуля или минимальной глубины и предавая глубины до тех пор, пока не будет найдено решение.

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

Аргумент problem представляет собой задачу, которую нужно решить. Это может быть описание начального состояния и правил перехода к новым состояниям.

14. Какова роль переменной limit в алгоритме?

Переменная limit определяет максимальную глубину поиска в текущем вызове. Она управляет пределами, до которых исследуются узлы на текущем этапе.

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

Это означает, что цикл будет проходить через все возможные целые числа, начиная с 1 и заканчивая максимально допустимым. Это позволяет исследовать глубины до тех пор, пока не будет найдено решение или не достигнуто максимальное значение.

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

Постепенное увеличение позволяет сначала исследовать более мелкие глубины, потенциально находя решения быстрее, и избегать ненужных затрат по времени и ресурсам в случае, если решение находится близко к корневому узлу.

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

Внутри цикла вызывается функция depth_limited_search, которая отвечает за поиск решения на текущем уровне глубины.

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

Функция depth_limited_search выполняет поиск на заданной глубине и возвращает найденное решение или специальный сигнал о том, что решение не было найдено на данном уровне.

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

cutoff обозначает предел глубины, в пределах которого будет производиться поиск на данном этапе. Если поиск достигает этого предела, он завершается или возвращает специальное значение, показывающее, что решение не найдено.

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

Это необходимо для определения, было ли найдено решение в рамках допустимой глубины. Если результат совпадает с cutoff, это значит, что все возможные узлы в пределах этой глубины были исследованы, но решение не найдено.

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

Если решение будет найдено на первой итерации, алгоритм завершится, и оно будет возвращено как результат, что позволяет быстро находить решения в случае, если они находятся на поверхности.

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

Функция может продолжать выполнение, чтобы проверить все возможные уровни глубины, до тех пор пока не достигнет максимально допустимого значения, что теоретически обеспечивает полный поиск, но на практике зависит от структуры задач.

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

Основные преимущества включают меньшую потребность в памяти, более высокую вероятность нахождения решения на меньших глубинах и возможность проверки всех уровней, что делает его более универсальным.

24. Какие потенциальные недостатки может иметь этот подход?

Недостатки могут включать завышенные временные затраты из за необходимости повторно исследовать узлы на каждом новом уровне и потенциально высокие временные затраты при глубоком дереве с высоким коэффициентом разветвления.