

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»

Выполнил:
Говоров Егор Юрьевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение
средств вычислительной
техники и
автоматизированных систем»,
очная форма обучения

(подпись)

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

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

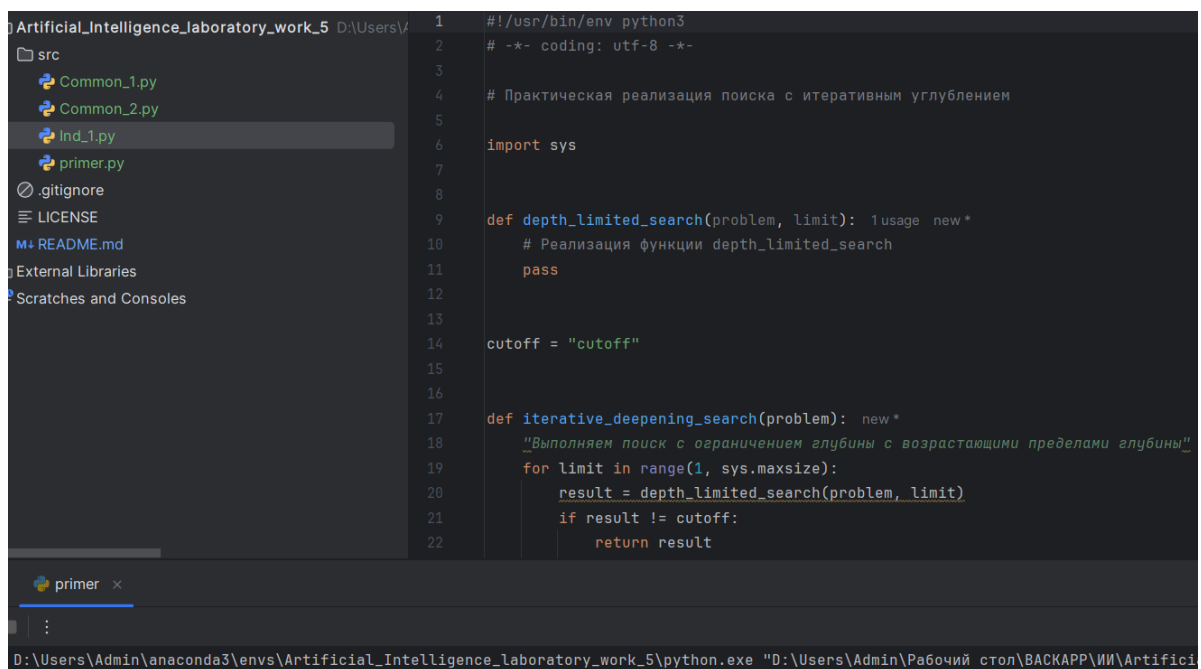
Тема: Исследование поиска с итеративным углублением

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

Ссылка на гитхаб: https://github.com/Artorias1469/Artificial_Intelligence_1_laboratory_work_5.git

Ход работы:

1. Проработал примеры из лабораторной работы:



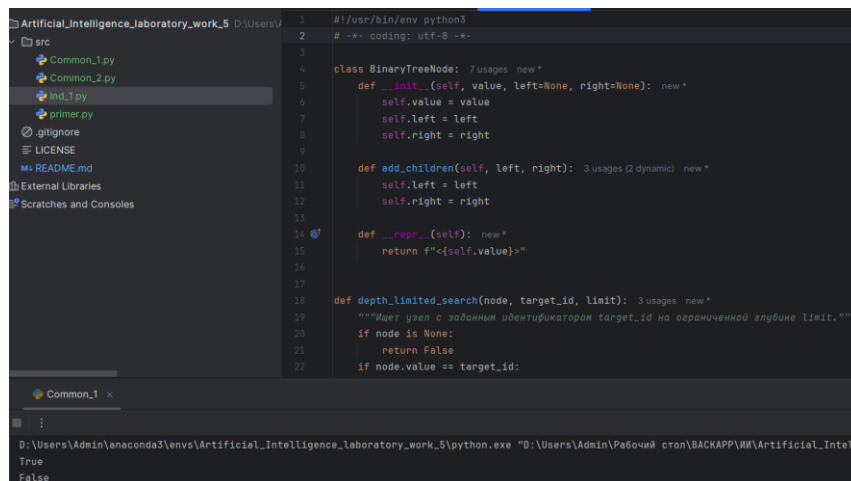
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files: Common_1.py, Common_2.py, Ind_1.py, primer.py, .gitignore, LICENSE, README.md, External Libraries, and Scratches and Consoles. The code editor shows the following Python code:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # Практическая реализация поиска с итеративным углублением
5
6  import sys
7
8
9  def depth_limited_search(problem, limit):
10     # Реализация функции depth_limited_search
11     pass
12
13
14     cutoff = "cutoff"
15
16
17  def iterative_deepening_search(problem):
18     """Выполняем поиск с ограничением глубины с возрастающими пределами глубины"""
19     for limit in range(1, sys.maxsize):
20         result = depth_limited_search(problem, limit)
21         if result != cutoff:
22             return result
```

Рисунок 1. Выполнение примера из ЛР

2. Выполнение общих заданий:

1. Представьте себе систему управления доступом, где каждый пользователь представлен узлом в дереве. Каждый узел содержит уникальный идентификатор пользователя. Ваша задача — разработать метод поиска, который позволит проверить существование пользователя с заданным идентификатором в системе, используя структуру дерева и алгоритм итеративного углубления.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 class BinaryTreeNode: 7 usages new *
5     def __init__(self, value, left=None, right=None): new *
6         self.value = value
7         self.left = left
8         self.right = right
9
10    def add_children(self, left, right): 3 usages (2 dynamic) new *
11        self.left = left
12        self.right = right
13
14    def __repr__(self): new *
15        return f"<{self.value}>"
16
17
18    def depth_limited_search(node, target_id, limit): 3 usages new *
19        """Ищет узел с заданным идентификатором target_id на ограниченной глубине limit."""
20        if node is None:
21            return False
22        if node.value == target_id:
```

Common_1 x

D:\Users\Admin\anaconda3\envs\Artificial_Intelligence_laboratory_work_5\python.exe "D:\Users\Admin\Рабочий стол\БАСКАПП\ИИ\Artificial_Intell

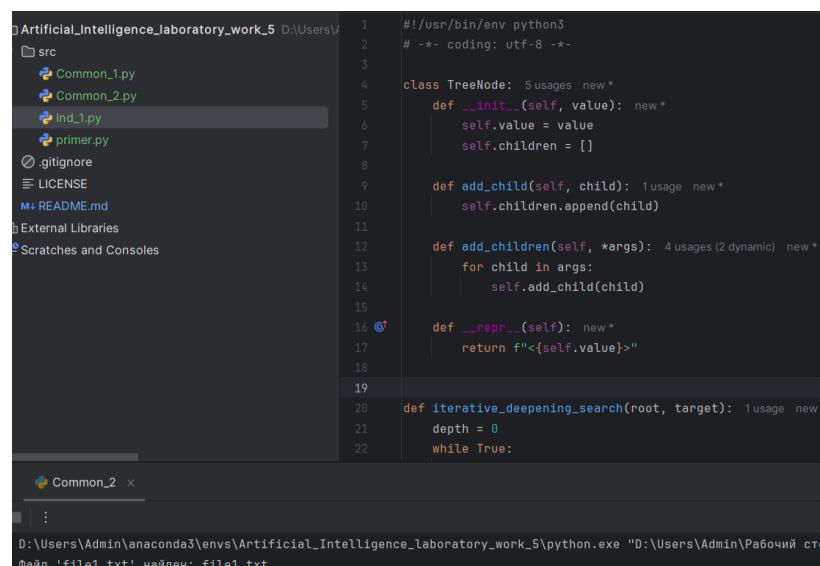
True

False

Рисунок 2. Результат выполнения первого общего задания

2. 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.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 class TreeNode: 5 usages new *
5     def __init__(self, value): new *
6         self.value = value
7         self.children = []
8
9     def add_child(self, child): 1 usage new *
10        self.children.append(child)
11
12    def add_children(self, *args): 4 usages (2 dynamic) new *
13        for child in args:
14            self.add_child(child)
15
16    def __repr__(self): new *
17        return f"<{self.value}>"
18
19
20    def iterative_deepening_search(root, target): 1 usage new *
21        depth = 0
22        while True:
```

Common_2 x

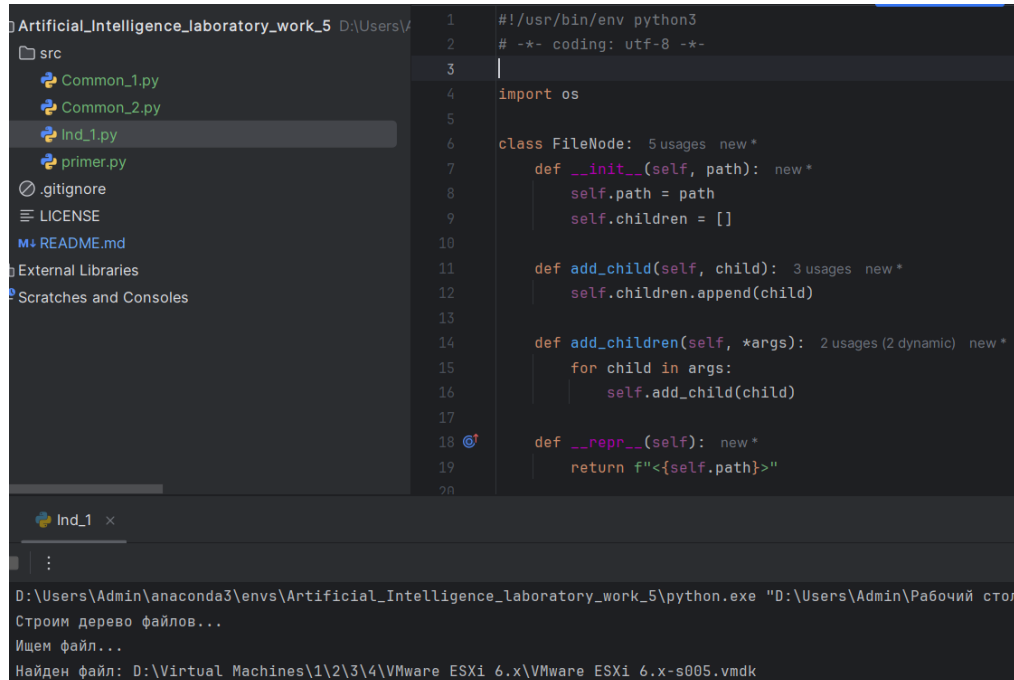
D:\Users\Admin\anaconda3\envs\Artificial_Intelligence_laboratory_work_5\python.exe "D:\Users\Admin\Рабочий ст

Файл 'file1.txt' найден: file1.txt

Рисунок 3. Результат выполнения второго общего задания

3. Выполнение индивидуального задания:

Поиск по размеру файла. В директориях существуют файлы различного размера. Найти первый файл размером более 1 ГБ, который находится на глубине не менее 5 уровней.



The screenshot shows a Python IDE with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like Common_1.py, Common_2.py, Ind_1.py, primer.py, .gitignore, LICENSE, README.md, External Libraries, and Scratches and Consoles. The code editor shows a Python script for finding files by size and depth. The script defines a class FileNode and methods for adding children and finding files. The output window at the bottom shows the execution of the script, which finds a file named D:\Virtual Machines\1\2\3\4\VMware ESXi 6.x\VMware ESXi 6.x-s005.vmdk.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import os
5
6  class FileNode:
7      def __init__(self, path):
8          self.path = path
9          self.children = []
10
11      def add_child(self, child):
12          self.children.append(child)
13
14      def add_children(self, *args):
15          for child in args:
16              self.add_child(child)
17
18      def __repr__(self):
19          return f"<{self.path}>"
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Ind_1 x

D:\Users\Admin\anaconda3\envs\Artificial_Intelligence_laboratory_work_5\python.exe "D:\Users\Admin\Рабочий стол

Строим дерево файлов...

Ищем файл...

Найден файл: D:\Virtual Machines\1\2\3\4\VMware ESXi 6.x\VMware ESXi 6.x-s005.vmdk

Рисунок 4. Результат выполнения индивидуального задания

Ответы на контрольные 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 определяет текущую глубину узла. Он влияет на поиск, задавая ограничение на количество уровней, которые может пройти алгоритм, предотвращая исследование узлов, выходящих за пределы этой глубины.

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

Оптимальное значение глубины d зависит от неизвестной заранее глубины решения. Если d слишком мало, решение может быть пропущено; если слишком велико, алгоритм может потреблять лишние ресурсы или изучать ненужные пути.

3. Какие преимущества дает использование алгоритма итеративного

углубления по сравнению с поиском в ширину?

Итеративное углубление использует меньше памяти, так как оно работает как поиск в глубину, при этом сохраняя полноту и оптимальность поиска в ширину, если стоимости переходов равны.

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

Алгоритм выполняет поиск в глубину с увеличением лимита глубины на каждой итерации. Это помогает избежать экспоненциального роста памяти, характерного для поиска в ширину, так как для поиска в глубину требуется только стек глубины.

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

Начинать поиск заново необходимо, потому что алгоритм в предыдущей итерации не сохраняет узлы, которые превышают текущий лимит глубины. Это позволяет экономить память.

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

Временная сложность: $O(bd)$, где b — коэффициент разветвления, d — глубина решения. Пространственная сложность: $O(d)$, так как требуется память только для стека глубины.

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

Он использует память, как поиск в глубину, но сохраняет полноту и оптимальность, как поиск в ширину, постепенно исследуя узлы на увеличивающихся уровнях глубины.

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

Большая часть времени уходит на исследование узлов на самом

глубоком уровне, так как число узлов растет экспоненциально. Повторное исследование более верхних уровней имеет относительно низкую стоимость.

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

Алгоритм генерирует порядка bd узлов. Однако за счет повторения на малых глубинах суммарное количество сгенерированных узлов составляет $O(b^d)$.

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

Если генерация узлов на верхних уровнях дерева требует значительных вычислений, то повторное их исследование может быть дорогостоящим. Также он неэффективен, если стоимость переходов не одинакова.

11. Какую задачу решает функция `iterative_deepening_search`?

Она находит решение задачи методом итеративного углубления, возвращая путь к решению или указание на то, что решение отсутствует.

12. Каков основной принцип работы поиска с итеративным углублением?

Постепенное увеличение предела глубины и выполнение поиска в глубину на каждом уровне до тех пор, пока не будет найдено решение.

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

Это объект, описывающий задачу поиска, включающий начальное состояние, функцию определения целей и правила переходов.

14. Какова роль переменной `limit` в алгоритме?

Она задает текущую максимальную глубину поиска.

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

Диапазон задает последовательное увеличение предела глубины с минимального значения до максимально возможного в системе.

16. Почему предел глубины поиска увеличивается постепенно, а не

устанавливается сразу на максимальное значение?

Это позволяет найти решение на минимальной глубине, избегая ненужных затрат ресурсов на более глубокие уровни.

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

Функция `depth_limited_search` выполняет поиск с ограничением глубины, проверяя, можно ли найти решение в пределах текущей глубины.

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

Она проверяет узлы до заданной глубины. Возвращает либо путь к решению, либо указание, что поиск был "обрезан" (`cutoff`), либо "неудачу".

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

`Cutoff` обозначает, что поиск достиг текущего предела глубины и не смог продолжить исследование.

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

Это позволяет алгоритму определить, нужно ли увеличить глубину на следующей итерации.

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

Алгоритм завершится, вернув найденное решение.

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

Это позволяет алгоритму исследовать все возможные уровни глубины, если решение расположено на большом расстоянии.

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

Он сохраняет полноту и оптимальность, чего не может гарантировать обычный поиск в глубину.

24. Какие потенциальные недостатки может иметь этот подход?

Повторное исследование узлов может быть дорогостоящим в терминах времени, особенно если дерево большое или генерация узлов сложна.

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

- Использование эвристик для пропуска заведомо нерелевантных узлов.
- Применение более эффективного способа хранения и повторного использования верхних уровней дерева.

Вывод: приобрел навыки по работе с поиском с итеративным углублением с помощью языка программирования Python версии 3.x