

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 1

Выполнил:
Бабенко Артём Тимофеевич
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:
Ассистент департамента цифровых,
робототехнических систем и
электроники Богданов С.С

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

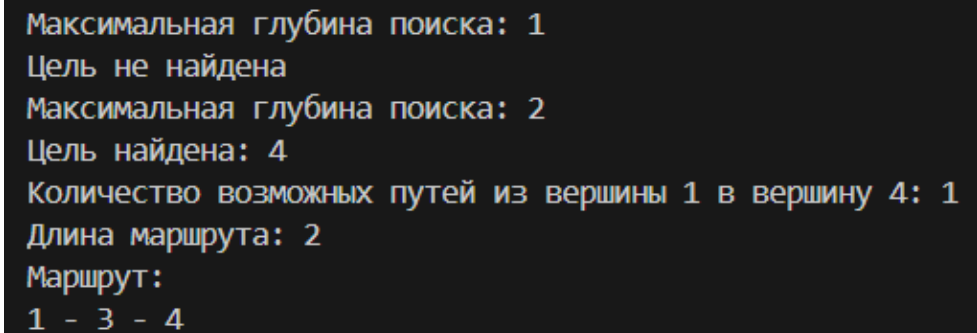
Тема: Исследование поиска с итеративным углублением

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

Порядок выполнения работы:

Задание 1. Поиск элемента в дереве с использованием алгоритма итеративного углубления.

Результат работы программы:



```
Максимальная глубина поиска: 1
Цель не найдена
Максимальная глубина поиска: 2
Цель найдена: 4
Количество возможных путей из вершины 1 в вершину 4: 1
Длина маршрута: 2
Маршрут:
1 - 3 - 4
```

Рисунок 1 – Результат работы программы

Код программы:

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

from collections import defaultdict, deque, Counter
from itertools import combinations
from collections import deque
import sys

class Node:
    "Узел в дереве поиска"
    def __init__(self, state, parent=None, action=None, path_cost=0.0):
        self.__dict__.update(state=state, parent=parent, action=action,
                               path_cost=path_cost)
    def __repr__(self): return '<{}>'.format(self.state)
    def __len__(self): return 0 if self.parent is None else (1 + len(self.parent))
    def __lt__(self, other): return self.path_cost < other.path_cost
    def cost_calc(self, matrix):

        return matrix[self.action[len(self.action)-2]][self.action[len(self.action)-1]]

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 expand(node, max_depth = 10**27, finish = 5):
    "Раскрываем узел, создав дочерние узлы."
    s = node.state
    last_node = node.action
    #print(last_node)
    #print(tree[int(last_node)])
    if node.path_cost <= max_depth:
        lst_nodes = [n for n in [last_node.left, last_node.right] if n != None]
        result = []
        for item in lst_nodes:
            temp = s.copy()
            if item not in s:
                temp.append(item)
                dist = node.path_cost + 1
                result.append(Node(temp, node, item, dist))

        return result
    else:
        return []

if __name__ == '__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))

    start = root.value
    max_depth = 1
    finish = 4
    paths = []
    find = False
    while max_depth < sys.maxsize and find != True:
        print(f'Максимальная глубина поиска: {max_depth}')
        first = Node([root], None, root, 0)
        paths = []
        q = deque()
        q.appendleft(first)
        counter = 0
        while q:
            counter += 1
            temp = expand(q.popleft(), max_depth - 1)
            for i in temp:

```


Код программы:

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

from collections import defaultdict, deque, Counter
from itertools import combinations
from collections import deque
import sys

class Node:
    "Узел в дереве поиска"
    def __init__(self, state, parent=None, action=None, path_cost=0.0):
        self.__dict__.update(state=state, parent=parent, action=action,
                               path_cost=path_cost)
    def __repr__(self): return '<{}>'.format(self.state)
    def __len__(self): return 0 if self.parent is None else (1 + len(self.parent))
    def __lt__(self, other): return self.path_cost < other.path_cost
    def cost_calc(self, matrix):

        return matrix[self.action[len(self.action)-2]][self.action[len(self.action)-1]]

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 expand(node, max_depth = 10**27, finish = 5):
    "Раскрываем узел, создав дочерние узлы."
    s = node.state
    last_node = node.action
    #print(last_node)
    #print(tree[int(last_node)])
    if node.path_cost <= max_depth:
        lst_nodes = [n for n in last_node.children if n != None]
        result = []
        for item in lst_nodes:
            temp = s.copy()
            if item not in s:
                temp.append(item)
                dist = node.path_cost + 1
                result.append(Node(temp,node, item, dist))
```

[illegible]

```

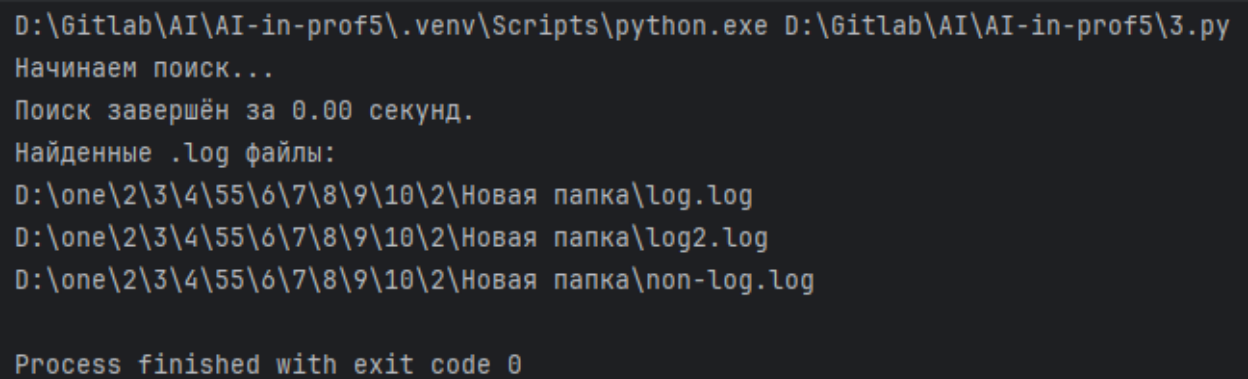
mn[1] = i.state
#print(i.state,i.path_cost,counter)
counter+=1

print(f'Количество возможных путей из вершины {start} в вершину {finish}:
{len(paths)}")
print(f'Длина маршрута: {mn[0]}")
print(f'Маршрут:')
for id in mn[1]:
    if id.value!= finish:
        print(id.value,end=" --> ")
    else:
        print(id.value)

```

Задание 3. 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.

Результат работы программы:



```

D:\Gitlab\AI\AI-in-prof5\.venv\Scripts\python.exe D:\Gitlab\AI\AI-in-prof5\3.py
Начинаем поиск...
Поиск завершён за 0.00 секунд.
Найденные .log файлы:
D:\one\2\3\4\55\6\7\8\9\10\2\Новая папка\log.log
D:\one\2\3\4\55\6\7\8\9\10\2\Новая папка\log2.log
D:\one\2\3\4\55\6\7\8\9\10\2\Новая папка\non-log.log

Process finished with exit code 0

```

Рисунок 3 – Результат работы программы

Код программы:

```

import os
import time
from concurrent.futures import ThreadPoolExecutor

def find_log_files_multithreaded(root_dir):
    log_files = []

    def process_directory(directory):
        try:

```

```

with os.scandir(directory) as entries:
    for entry in entries:
        if entry.is_file() and entry.name.lower().endswith('.log'):
            log_files.append(entry.path)
        elif entry.is_dir():
            process_directory(entry.path)
except PermissionError:
    print(f"Нет доступа к директории: {directory}")
except FileNotFoundError:
    print(f"Директория не существует: {directory}")
except OSError as e:
    print(f"Ошибка при обработке директории '{directory}': {e}")

with ThreadPoolExecutor(max_workers=8) as executor: # 8 потоков
    executor.submit(process_directory, root_dir)

return log_files

```

Пример использования

```

if __name__ == "__main__":
    root_directory = r"D:\one\2\3\4\55\6\7\8\9\10\2\Новая папка"

    if os.path.exists(root_directory) and os.path.isdir(root_directory):
        print("Начинаем поиск...")
        start_time = time.time()

        log_files = find_log_files_multithreaded(root_directory)

        end_time = time.time()
        print(f"Поиск завершён за {end_time - start_time:.2f} секунд.")

    print("Найденные .log файлы:")
    for file in log_files:
        print(file)

```



```
else:  
    print(f"Указанный путь '{root_directory}' не существует или не является  
директорией.")
```

Ответы на контрольные вопросы:

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

n — максимальная глубина поиска, это значение влияет на глубину поиска.

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

В большинстве случаев информация о глубине решения отсутствует.

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

Это позволяет снизить затраты памяти.

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

Для каждой глубины алгоритм начинает поиск заново с корневого узла, что позволяет избежать проблем с памятью, связанных с хранением всех узлов на более глубоких уровнях. Преимущество итеративного углубления в том, что оно использует небольшое количество памяти, так как хранит только узлы текущего уровня и узлы предшествующих уровней (в отличие от поиска в ширину, который хранит все узлы на текущем уровне). Это позволяет эффективно использовать память, особенно в больших деревьях.

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

Алгоритм итеративного углубления начинает поиск заново с корневого узла на каждой итерации по нескольким причинам:

- 1) Неопределенность решения.
- 2) Избежание бесконечных циклов.

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

Итеративное углубление имеет временную сложность $O(b^d)$, где b — это коэффициент ветвления, а d — глубина решения. Это связано с тем, что на каждой итерации выполняется полный поиск в глубину до текущей глубины.

Пространственная сложность составляет $O(b * d)$, что значительно меньше, чем у поиска в ширину ($O(b^d)$), поскольку хранится только стек текущего уровня и узлы предыдущих уровней.

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

Итеративное углубление сочетает в себе преимущества обоих подходов:

Поиск в глубину: Позволяет глубже исследовать возможные решения, что может быть полезно, если решение находится на большой глубине.

Поиск в ширину: Обеспечивает гарантии нахождения кратчайшего пути к решению, поскольку каждая новая глубина охватывает все узлы предыдущих уровней.

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

Хоть алгоритм и требует повторного вычисления узлов, его эффективность обеспечивается за счет оптимизации использования памяти и структуры дерева.

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

Коэффициент разветвления b и глубина d влияют на общее количество узлов следующим образом: $O(b^d)$

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

В случаях, когда стоимость шагов не одинакова.

11. Какую задачу решает функция `iterative_deepening_search` ? 12. Каков основной принцип работы поиска с итеративным углублением?

Данная функция на каждой итерации использует поиск с ограничением глубины, а потом увеличивает глубину.

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

`problem` представляет собой задачу, для которой необходимо найти решение.

14. Какова роль переменной `limit` в алгоритме?

`limit` – ограничение глубины поиска на каждой итерации.

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

Данный диапазон представляет из себя множество значений допустимой глубины на каждой итерации.

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

Потому что решение может быть на небольшой глубине.

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

Внутри цикла вызывается функция поиска с ограничением глубины для поиска решения на текущей глубине.

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

Данная функция ищет нужное значение на заданной глубине и возвращает результат или специальное значение в случае неудачи.

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

Это специальное значение, которое возвращается в случае неудачи.

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

Чтобы проверить, найден ли нужный элемент.

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

Будет возвращено значение, а цикл завершится.

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

Такое может произойти в случае отсутствия нужного элемента.

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

Временная сложность у данного поиска сопоставима с поиском в ширину.

24. Какие потенциальные недостатки может иметь этот подход?

Может генерироваться большее количество узлов, а в случае наличия путей с разной стоимостью работа алгоритма будет ухудшаться.

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

Данный алгоритм можно улучшить с помощью использования каких-либо эвристик.

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