

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 №4
дисциплины «Искусственный интеллект в профессиональной сфере»
Вариант ____

Выполнил:
Епифанов Алексей Александрович
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 Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: исследование поиска с ограничением глубины

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

Порядок выполнения работы:

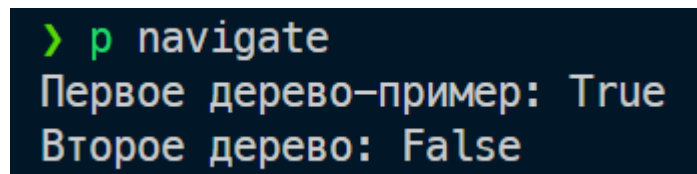
1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

Репозиторий: https://github.com/alexeiepif/ii_4.git

2. Выполнил задание: Система навигации робота-пылесоса.

Вы работаете над разработкой системы навигации для робота-пылесоса. Робот способен передвигаться по различным комнатам в доме, но из-за ограниченности ресурсов (например, заряда батареи) и времени на уборку, важно эффективно выбирать путь. Ваша задача - реализовать алгоритм, который поможет роботу определить, существует ли путь к целевой комнате, не превышая заданное ограничение по глубине поиска.

Дано дерево, где каждый узел представляет собой комнату в доме. Узлы связаны в соответствии с возможностью перемещения робота из одной комнаты в другую. Необходимо определить, существует ли путь от начальной комнаты (корень дерева) к целевой комнате (узел с заданным значением), так, чтобы робот не превысил лимит по глубине перемещения.



```
> p navigate
Первое дерево-пример: True
Второе дерево: False
```

Рисунок 1. Вывод программы для примера и другого дерева

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

```
from tree import Problem
from tree import depth_limited_search as dls
```

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

    def __repr__(self):
```

```

        return f"<{self.value}>"

class NavigateProblem(Problem):
    def __init__(self, initial, goal):
        super().__init__(initial, goal)

    def actions(self, state):
        left = state.left
        right = state.right
        if left:
            yield left
        if right:
            yield right

    def result(self, state, action):
        return action

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

def solve(root, goal, limit):
    problem = NavigateProblem(root, goal)
    r = dls(problem, limit)
    return bool(r)

if __name__ == "__main__":
    root = BinaryTreeNode(
        1,
        BinaryTreeNode(2, None, BinaryTreeNode(4)),
        BinaryTreeNode(3, BinaryTreeNode(5), None),
    )
    goal = 4
    limit = 2
    print("Первое дерево-пример:", solve(root, goal, limit))

    root2 = BinaryTreeNode(
        1,
        BinaryTreeNode(
            2,
            BinaryTreeNode(4, None, BinaryTreeNode(8)),
            BinaryTreeNode(5),
        ),
        BinaryTreeNode(
            3,
            BinaryTreeNode(6),
            BinaryTreeNode(7, None, BinaryTreeNode(9)),
        ),
    )
    goal = 8
    limit = 2
    print("Второе дерево:", solve(root2, goal, limit))

```

3. Выполнил задание: Система управления складом.

Представьте, что вы разрабатываете систему для управления складом, где товары упорядочены в структуре, похожей на двоичное дерево. Каждый узел дерева представляет место хранения, которое может вести к другим

местам хранения (левому и правому подразделу). Ваша задача — найти наименее затратный путь к товару, ограничив поиск заданной глубиной, чтобы гарантировать, что поиск займет приемлемое время.

```
> p stock
Первое дерево-пример. Цель найдена: <4>
Второе дерево: Цель не найдена
```

Рисунок 2. Вывод программы для примера и другого дерева

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

from tree import Problem
from tree import depth_limited_search as dls

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

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

class StockProblem(Problem):
    def __init__(self, initial, goal):
        super().__init__(initial, goal)

    def actions(self, state):
        left = state.left
        right = state.right
        if left:
            yield left
        if right:
            yield right

    def result(self, state, action):
        return action

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

def solve(root, goal, limit):
    problem = StockProblem(root, goal)
    r = dls(problem, limit)
    return r

if __name__ == "__main__":
    root = BinaryTreeNode(
        1,
        BinaryTreeNode(2, None, BinaryTreeNode(4)),
        BinaryTreeNode(3, BinaryTreeNode(5), None),
    )
```

```

goal = 4
limit = 2
r = solve(root, goal, limit)
if r:
    st = f"Цель найдена: {r.state}"
else:
    st = "Цель не найдена"
print("Первое дерево-пример. ", st)

root2 = BinaryTreeNode(
    1,
    BinaryTreeNode(
        2,
        BinaryTreeNode(4, None, BinaryTreeNode(8)),
        BinaryTreeNode(5),
    ),
    BinaryTreeNode(
        3,
        BinaryTreeNode(6),
        BinaryTreeNode(7, None, BinaryTreeNode(9)),
    ),
)
goal = 8
limit = 2
r = solve(root2, goal, limit)
if r:
    st = f"Цель найдена: {r.state}"
else:
    st = "Цель не найдена"
print("Второе дерево: ", st)

```

4. **Выполнил задание:** Система автоматического управления инвестициями.

Представьте, что вы разрабатываете систему для автоматического управления инвестициями, где дерево решений используется для представления последовательности инвестиционных решений и их потенциальных исходов. Цель состоит в том, чтобы найти наилучший исход (максимальную прибыль) на определённой глубине принятия решений, учитывая ограниченные ресурсы и время на анализ.

```

> p invest
Первое дерево-пример. Максимальное значение на указанной глубине: 6
Второе дерево: Максимальное значение на указанной глубине: 12

```

Рисунок 3. Вывод программы для примера и другого дерева

```

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

```

```

from tree import LIFOQueue, Node, Problem, expand, is_cycle

```

```

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

```

```

        self.value = value
        self.left = left
        self.right = right

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

class InvestProblem(Problem):
    def __init__(self, initial):
        super().__init__(initial)

    def actions(self, state):
        left = state.left
        right = state.right
        if left:
            yield left
        if right:
            yield right

    def result(self, state, action):
        return action

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

def dls(problem, limit=10):
    """В первую очередь ищем самые глубокие узлы в дереве поиска."""
    frontier = LIFOQueue([Node(problem.initial)])
    result = []
    while frontier:
        node = frontier.pop()
        if problem.is_goal(node.state):
            return node
        elif len(node) == limit:
            result.append(node.state.value)
        elif not is_cycle(node):
            for child in expand(problem, node):
                frontier.append(child)
    return max(result or [0])

def solve(root, limit):
    problem = InvestProblem(root)
    r = dls(problem, limit)
    return r

if __name__ == "__main__":
    root = BinaryTreeNode(
        3,
        BinaryTreeNode(1, BinaryTreeNode(0), None),
        BinaryTreeNode(5, BinaryTreeNode(4), BinaryTreeNode(6)),
    )
    limit = 2
    r = solve(root, limit)
    if r:
        st = f"Максимальное значение на указанной глубине: {r}"
    else:
        st = "Дерево не настолько глубокое"
    print("Первое дерево-пример. ", st)

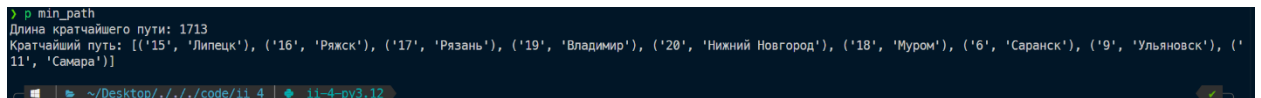
```

```

root2 = BinaryTreeNode(
    19,
    BinaryTreeNode(
        11,
        BinaryTreeNode(16, None, BinaryTreeNode(10)),
        BinaryTreeNode(15),
    ),
    BinaryTreeNode(
        6,
        BinaryTreeNode(18),
        BinaryTreeNode(13, None, BinaryTreeNode(12)),
    ),
)
goal = 8
limit = 3
r = solve(root2, limit)
if r:
    st = f"Максимальное значение на указанной глубине: {r}"
else:
    st = "Дерево не настолько глубокое"
print("Второе дерево: ", st)

```

5. Выполнил задание: Для построенного графа лабораторной работы 1 (имя файла начинается с PR.AI.001.) напишите программу на языке программирования Python, которая с помощью алгоритма поиска с ограничением глубины находит минимальное расстояние между начальным и конечным пунктами. Сравните найденное решение с решением, полученным вручную.



```

> p min_path
Длина кратчайшего пути: 1713
Кратчайший путь: [['15', 'Липецк'], ['16', 'Рязань'], ['17', 'Рязань'], ['19', 'Владимир'], ['20', 'Нижегород'], ['18', 'Муром'], ['6', 'Саранск'], ['9', 'Ульяновск'], ['11', 'Самара']]

```

Рисунок 4. Результат

```

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

import json

from tree import Problem, depth_limited_search, path_states

class MinPathProblem(Problem):
    def __init__(self, initial, goal, nodes, edges):
        super().__init__(initial, goal)
        self.nodes = nodes
        self.edges = edges

    def actions(self, state):
        for edge in self.edges:
            if edge["data"]["source"] == state[0] or edge["data"]["target"] == state[0]:
                yield edge["data"]

    def result(self, state, action):
        if state[0] == action["source"]:
            target = action["target"]

```

```

    else:
        target = action["source"]
    for node in self.nodes:
        if node["data"]["id"] == target:
            return (node["data"]["id"], node["data"]["label"])

def is_goal(self, state):
    return state == self.goal

def action_cost(self, s, a, s1):
    return a["weight"]

def load_elems(path):
    with open(path) as f:
        elems = json.load(f)

    for i, elem in enumerate(elems):
        if not elem.get("position", None):
            nodes_index = i
            break

    nodes = elems[:nodes_index]
    edges = elems[nodes_index:]
    return nodes, edges

def search(problem: Problem):
    limit = 8
    b = depth_limited_search(problem, limit)
    length = b.path_cost
    path = path_states(b)
    return length, path

def solve(init, goal, nodes, edges):
    problem = MinPathProblem(init, goal, nodes, edges)
    return search(problem)

if __name__ == "__main__":
    nodes, edges = load_elems("elem_full.json")
    for node in nodes:
        if node["data"]["label"] == "Липецк":
            initial = (node["data"]["id"], node["data"]["label"])
        elif node["data"]["label"] == "Самара":
            goal = (node["data"]["id"], node["data"]["label"])
    length, path = solve(initial, goal, nodes, edges)
    print("Длина кратчайшего пути:", length)
    print("Кратчайший путь:", path)

```

Полученный путь не оптимален, так как есть путь короче - 885.2км.

Так получилось, потому что поиск с ограничением глубины не предназначен для поиска минимального пути, он находит случайный путь указанной или меньшей глубины, и сразу возвращает его.

Ответы на контрольные вопросы:

1. Что такое поиск с ограничением глубины, и как он решает проблему бесконечных ветвей?

Стратегия поиска с ограничением глубины — это модификация поиска в глубину. Она предусматривает исследование дерева лишь до определённого уровня глубины.

Этот метод решает проблему бесконечной глубины m , возникающую, когда дерево расширяется бесконечно. Если решение существует на глубине, не превышающей максимально установленный уровень n , оно будет найдено. Поскольку ветвление дерева ограничено этой глубиной, исследование каждой ветви не будет продолжаться бесконечно. Таким образом, решается проблема бесконечных ветвей.

2. Какова основная цель ограничения глубины в данном методе поиска?

Избавление от проблемы бесконечных ветвей.

3. В чем разница между поиском в глубину и поиском с ограничением глубины?

Поиск в глубину не ограничен глубиной, и рассматривает все дерево целиком, а поиск с ограниченной глубиной рассматривает лишь часть дерева до этой глубины.

4. Какую роль играет проверка глубины узла в псевдокоде поиска с ограничением глубины?

Псевдокод этой стратегии указывает, что при спуске по дереву для поиска в глубину используется информация о глубине узла, записанная в его состоянии. Если глубина не превышает максимально допустимый уровень, происходит расширение узла, как обычно. В противном случае процесс обрывается, и осуществляется возврат к предыдущим узлам, после чего начинается исследование других ветвей.

5. Почему в случае достижения лимита глубины функция возвращает «обрезание»?

Функция возвращает обрезание, так как достигнута максимальная глубина, а решение еще не найдено.

6. В каких случаях поиск с ограничением глубины может не найти решение, даже если оно существует?

В случае, если решение расположено глубже лимита, данный алгоритм его не найдет.

7. Как поиск в ширину и в глубину отличаются при реализации с использованием очереди?

Если поиск по дереву реализуется с помощью очереди, основное отличие между поиском в глубину и ширину заключается в порядке добавления новых узлов в очередь.

8. Почему поиск с ограничением глубины не является оптимальным?

Поиск с ограниченной глубиной не является оптимальным, так как он может находить решения на глубинах n или $n-1$, в то время как более короткое решение может существовать на глубине 2, но оно не было достигнуто из-за ограничения глубины. Это делает поиск с ограниченной глубиной лишь немного лучше поиска в глубину.

9. Как итеративное углубление улучшает стандартный поиск с ограничением глубины?

Для улучшения поиска с ограниченной глубиной можно использовать стратегию итеративного углубления. Итеративное углубление сочетает в себе лучшие качества поиска в глубину и ширину. Оно итеративно увеличивает глубину поиска: начинается с глубины 1 и постепенно увеличивается. Такой подход обеспечивает полноту и оптимальность поиска в ширину, сохраняя при этом низкую пространственную сложность поиска в глубину.

10. В каких случаях итеративное углубление становится эффективнее простого поиска в ширину?

Итеративное углубление эффективно для деревьев с высоким коэффициентом ветвления, так как оно избегает хранения большого числа узлов в памяти, что является проблемой для поиска в ширину. Каждая

итерация увеличивает глубину, позволяя обнаруживать решения на меньших глубинах быстрее, чем поиск с фиксированной ограниченной глубиной, обеспечивая тем самым оптимальность результата.

11. Какова основная цель использования алгоритма поиска с ограничением глубины?

Проверить, существует ли решение до или на указанной глубине.

12. Какие параметры принимает функция `depth_limited_search` , и каково их назначение?

Функция `depth_limited_search` принимает два аргумента: `problem` (задача, которую нужно решить) и `limit` (максимальная глубина поиска).

13. Какое значение по умолчанию имеет параметр `limit` в функции `depth_limited_search` ?

10

14. Что представляет собой переменная `frontier`, и как она используется в алгоритме?

`Frontier` это `LIFOQueue`, хранящая раскрытые непроверенные узлы.

15. Какую структуру данных представляет `LIFOQueue` , и почему она используется в этом алгоритме?

Очередь «последним пришел – первым ушел»

16. Каково значение переменной `result` при инициализации, и что оно означает?

`Result` инициализируется узлом `failure`, который означает неудачу в поиске.

17. Какое условие завершает цикл `while` в алгоритме поиска?

Начинается цикл, который выполняется до тех пор, пока `frontier` не станет пустым, то есть пока есть узлы для рассмотрения.

18. Какой узел извлекается с помощью `frontier.pop()` и почему?

Извлекается последний добавленный узел из `frontier` для дальнейшей обработки. Последний – так как используется `lifo` очередь.

19. Что происходит, если найден узел, удовлетворяющий условию цели (условие `problem.is_goal(node.state)`)?

Проверяется, является ли текущий узел целевым. Если да, то поиск завершен успешно и возвращает текущий узел как результат успешного поиска.

20. Какую проверку выполняет условие `elif len(node) >= limit`, и что означает его выполнение?

Проверяется, достиг ли текущий узел ограничения по глубине. Если да, то дальнейший поиск в этом направлении прекращается.

21. Что произойдет, если текущий узел достигнет ограничения по глубине поиска?

Если текущий узел достиг ограничения по глубине, переменной `result` присваивается значение `cutoff`, что означает достижение лимита глубины поиска.

22. Какую роль выполняет проверка на циклы `elif not is_cycle(node)` в алгоритме?

Проверяется, не ведет ли текущий узел к циклу. Если нет, то можно продолжать поиск.

23. Что происходит с дочерними узлами, полученными с помощью функции `expand(problem, node)`?

Каждый дочерний узел добавляется в `frontier` для дальнейшей обработки.

24. Какое значение возвращается функцией, если целевой узел не был найден?

Если целевой узел так и не был найден, возвращается значение `result`, которое может быть либо `failure`, либо `cutoff`, в зависимости от результата поиска.

25. В чем разница между результатами `failure` и `cutoff` в контексте данного алгоритма?

Failure – решения нет; cutoff – алгоритм проверил все дерево до указанной глубины, но не нашел решения, хотя оно может существовать глубже.

Вывод: выполняя данную работу были получены навыки по работе с поиском с ограничением глубины с помощью языка программирования Python версии 3.x