

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 №4
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 ограничением глубины с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Задача 1. 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.

Листинг 1 – Код программы general_robot.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 __repr__(self):
        return f"<{self.value}>"

def depth_limited_search(node, goal, depth_limit):
    """Функция поиска с ограничением по глубине."""
    if node is None:
        return False

    if node.value == goal:
        return True
```

```

if depth_limit <= 0:
    return False

return (depth_limited_search(node.left, goal, depth_limit - 1) or
        depth_limited_search(node.right, goal, depth_limit - 1))

def main():
    root = BinaryTreeNode(
        1,
        BinaryTreeNode(2, None, BinaryTreeNode(4)),
        BinaryTreeNode(3, BinaryTreeNode(5), None)
    )

    goal = 4
    limit = 2

    found = depth_limited_search(root, goal, limit)
    print(f"Найден на глубине: {found}")

if __name__ == "__main__":
    main()

```

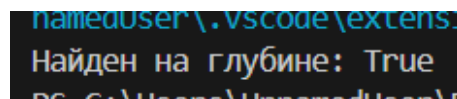


Рисунок 1 – Результат вывода программы

Задача 2. 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.

Листинг 2 – Код программы general_warehouse.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 __repr__(self):
    return f"<{self.value}>"

def depth_limited_search(node, goal, depth_limit):
    """Функция поиска с ограничением по глубине для нахождения цели."""
    if node is None:
        return None

    if node.value == goal:
        return node

    if depth_limit <= 0:
        return None

    left_result = depth_limited_search(node.left, goal, depth_limit - 1)
    if left_result:
        return left_result

    return depth_limited_search(node.right, goal, depth_limit - 1)

def main():
    root = BinaryTreeNode(
        1,
        BinaryTreeNode(2, None, BinaryTreeNode(4)),
        BinaryTreeNode(3, BinaryTreeNode(5), None)
    )

    goal = 4
    limit = 2

    result = depth_limited_search(root, goal, limit)
    if result:
        print(f"Цель найдена: {result}")
    else:
        print("Цель не найдена")

if __name__ == "__main__":
    main()

```

A terminal window with a dark background. The first line shows a prompt 'nameduser \\.vscode\' followed by a command. The second line shows the output 'Цель найдена: <4>'.

Рисунок 2 – Результат вывода программы

Задача 3. 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.

Листинг 3 – Код программы general_invest.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 __repr__(self):
        return f"<{self.value}>"

def find_max_at_depth(node, depth, current_depth=0):
    """Функция для нахождения максимального значения на указанной глубине."""
    if node is None:
        return float('-inf')

    if current_depth == depth:
        return node.value

    left_max = find_max_at_depth(node.left, depth, current_depth + 1)
    right_max = find_max_at_depth(node.right, depth, current_depth + 1)

    return max(left_max, right_max)

def main():
    root = BinaryTreeNode(
        3,
```

```

        BinaryTreeNode(1, BinaryTreeNode(0), None),
        BinaryTreeNode(5, BinaryTreeNode(4), BinaryTreeNode(6)),
    )

    limit = 2
    max_value = find_max_at_depth(root, limit)
    print(f"Максимальное значение на указанной глубине: {max_value}")

if __name__ == "__main__":
    main()

```

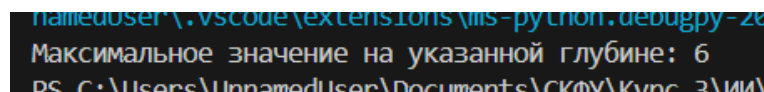


Рисунок 3 – Результат вывода программы

Индивидуальная задача. Для построенного графа лабораторной работы 1 напишите программу на языке программирования Python, которая с помощью алгоритма поиска с ограничением глубины находит минимальное расстояние между начальным и конечным пунктами. Определите глубину дерева поиска, на которой будет найдено решение.

Листинг 4 – Код программы individual.py

```

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

import itertools
import networkx as nx
import matplotlib.pyplot as plt

# Пример входных данных (можно заменить своими)
cities = {
    'Лонгфорд': {'Ньюнхем': 31.4, 'Экстон': 37.1, 'Бреона': 51.4, 'Конара': 43.2, 'Дерби': 111.9},
    'Конара': {'Сент-Мэрис': 73.9, 'Кэмпбелл-Таун': 12.5},
    'Кэмпбелл-Таун': {'Танбридж': 27.1, 'Лейк Лик': 34.8},
    'Лейк Лик': {'Бичено': 57, 'Суонси': 33.8},
    'Ньюнхем': {'Джордж Таун': 44.3, 'Лилидейл': 21.3},
    'Джордж Таун': {},
    'Лилидейл': {'Лебрина': 8.7},
    'Лебрина': {'Пайперс Брук': 13.3, 'Бридпорт': 27},
    'Экстон': {'Элизабет Таун': 18.4, 'Мол Крик': 30.8, 'Бреона': 38.4},
    'Элизабет Таун': {'Шеффилд': 28, 'Девонпорт': 42.5},
    'Девонпорт': {},
    'Шеффилд': {'Мойна': 31.7},

```

```

'Mойна': {},
'Бреона': {'Рейнольдс Лейк': 11.2, 'Шеннон': 26.5, 'Ботуэлл': 66.7},
'Рейнольдс Лейк': {'Миена': 18.5},
'Мол Крик': {'Шеффилд': 51.5},
'Миена': {'Тарралия': 59.2},
'Шеннон': {'Миена': 17.2},
'Тарралия': {'Уэйятина': 16.5},
'Уэйятина': {},
'Ботуэлл': {},
'Танбридж': {},
'Литл Суонпорт': {},
'Суонси': {'Литл Суонпорт': 27.7},
'Сент-Мэрис': {'Гарденс': 55.8},
'Гарденс': {'Дерби': 61.1},
'Дерби': {},
'Пайперс Брук': {},
'Бридпорт': {},
}

```

```

start_city = 'Гарденс' # Исходный город
end_city = 'Мойна' # Целевой город
max_depth = 6 # Ограничение глубины поиска

```

```

# Создание симметричного графа
def create_symmetric_graph(cities):
    symmetric_cities = {}
    for city, neighbors in cities.items():
        if city not in symmetric_cities:
            symmetric_cities[city] = {}
        for neighbor, distance in neighbors.items():
            symmetric_cities[city][neighbor] = distance
            if neighbor not in symmetric_cities:
                symmetric_cities[neighbor] = {}
            symmetric_cities[neighbor][city] = distance
    return symmetric_cities

```

```

symmetric_cities = create_symmetric_graph(cities)

```

```

# Генерация всех возможных маршрутов с ограничением глубины
def find_routes(cities, start, end, max_depth):
    routes = []

    def dfs(path, current_city, depth):
        if current_city == end:
            routes.append(path)
            return
        if depth >= max_depth:
            return
        for neighbor in cities.get(current_city, {}):

```

```

        if neighbor not in path:
            dfs(path + [neighbor], neighbor, depth + 1)

    dfs([start], start, 0)
    return routes

# Вычисление длины маршрута
def calculate_distance(route, graph):
    distance = 0
    for i in range(len(route) - 1):
        distance += graph[route[i]][route[i + 1]]
    return distance

# Построение графа и отображение маршрутов
def plot_graph(cities, routes, shortest_route):
    G = nx.DiGraph()

    # Добавление рёбер с весами
    for city, neighbors in cities.items():
        for neighbor, weight in neighbors.items():
            G.add_edge(city, neighbor, weight=weight)

    pos = nx.spring_layout(G) # Позиционирование узлов

    # Отображение графа
    plt.figure(figsize=(12, 8))
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw(G, pos, with_labels=True, node_size=700, node_color='lightblue', font_size=8)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

    # Подсветка всех маршрутов
    for route in routes:
        edges = [(route[i], route[i + 1]) for i in range(len(route) - 1)]
        nx.draw_networkx_edges(G, pos, edgelist=edges, edge_color='gray', width=1,
                               style='dotted')

    # Подсветка альтернативных маршрутов
    for route in routes:
        if route != shortest_route:
            alt_edges = [(route[i], route[i + 1]) for i in range(len(route) - 1)]
            nx.draw_networkx_edges(G, pos, edgelist=alt_edges, edge_color='blue', width=1)

    # Подсветка самого короткого маршрута
    shortest_edges = [(shortest_route[i], shortest_route[i + 1]) for i in range(len(shortest_route) - 1)]
    nx.draw_networkx_edges(G, pos, edgelist=shortest_edges, edge_color='red', width=2)

    plt.title("Граф маршрутов")
    plt.show()

```



```

# Основной код
all_routes = find_routes(symmetric_cities, start_city, end_city, max_depth)
all_routes_with_distances = [(route, calculate_distance(route, symmetric_cities)) for route in
all_routes]
sorted_routes = sorted(all_routes_with_distances, key=lambda x: x[1])

# Вывод всех маршрутов
print("Все маршруты из", start_city, "в", end_city, "с ограничением глубины", max_depth,
":")
for route, distance in sorted_routes:
    print(f"Маршрут: {' -> '.join(route)}, Расстояние: {round(distance, 1)} км")

# Вывод самого короткого маршрута
if sorted_routes:
    shortest_route = sorted_routes[0][0]
    shortest_distance = sorted_routes[0][1]
    print(f"\nСамый короткий маршрут: {' -> '.join(shortest_route)}, Расстояние:
{round(shortest_distance, 1)} км")
    plot_graph(symmetric_cities, [route for route, _ in sorted_routes], shortest_route)

```

При установленном значении максимальной глубины 7 программа находит 6 маршрутов (рисунки 4 и 5).

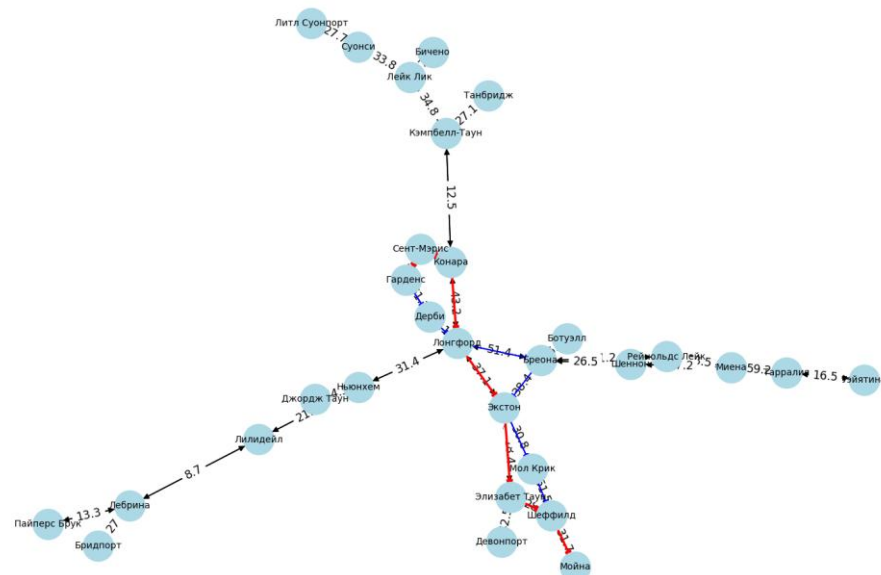


Рисунок 4 – Граф (максимальная глубина 7)

```

Все маршруты из Гарденс в Мойна с ограничением глубины 7 :
Маршрут: Гарденс -> Сент-Мэрис -> Коуара -> Лонгфорд -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 288.1 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 288.2 км
Маршрут: Гарденс -> Сент-Мэрис -> Коуара -> Лонгфорд -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 324.0 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 324.1 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Бреона -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 340.9 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Бреона -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 376.8 км

```

Рисунок 5 – Вывод программы (максимальная глубина 7)

Если максимальную глубину уменьшить до 6, программа выводит всего 2 варианта (рисунки 6 и 7).

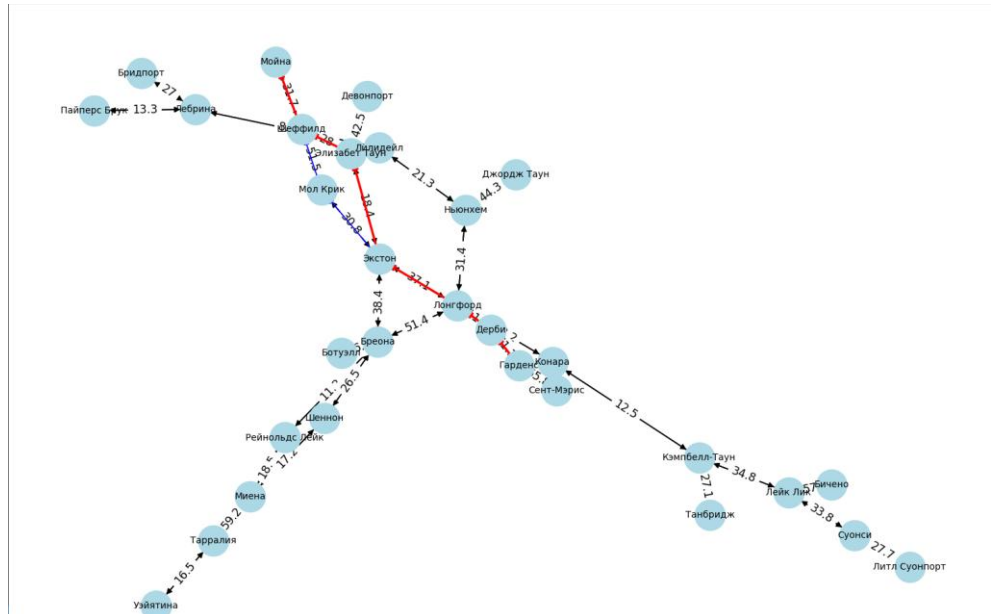


Рисунок 6 – Граф (максимальная глубина 6)

```

Все маршруты из Гарденс в Мойна с ограничением глубины 6 :
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Экстон -> Элизабет Таун -> Шеффилд -> Мойна, Расстояние: 288.2 км
Маршрут: Гарденс -> Дерби -> Лонгфорд -> Экстон -> Мол Крик -> Шеффилд -> Мойна, Расстояние: 324.1 км

```

Рисунок 7 – Вывод программы (максимальная глубина 6)

Ответы на контрольные вопросы:

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

Поиск с ограничением глубины — это алгоритм поиска, который ограничивает глубину обхода дерева или графа. Он решает проблему бесконечных ветвей, ограничивая количество уровней, на которых будет происходить поиск. Это предотвращает застревание в бесконечных ветвях

(например, циклических), позволяя алгоритму завершить работу после достижения заданной глубины.

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

Основная цель ограничения глубины — это ограничение ресурса поиска (например, глубины), чтобы контролировать время выполнения и предотвратить застревание в бесконечных или слишком глубоких ветвях. Это делает алгоритм более управляемым, особенно когда пространство поиска велико или может содержать циклы.

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

Разница между поиском в глубину и поиском с ограничением глубины состоит в том, что в поиске в глубину алгоритм продолжает углубляться в дереве до тех пор, пока не достигнет цели или не выйдет за пределы пространства поиска. В поиске с ограничением глубины существует жесткое ограничение на максимальную глубину, которое прерывает поиск, если оно достигнуто, не дождавшись нахождения цели.

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

Роль проверки глубины узла заключается в том, чтобы ограничить поиск. Каждый узел проверяется на соответствие глубине, и если глубина узла превышает заданный лимит, дальнейший обход не продолжается.

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

Возврат «обрезания» происходит, когда алгоритм достигает максимальной глубины (лимита), не найдя цели. Это означает, что данный

путь не будет продолжен, и алгоритм будет двигаться по другим возможным путям, но с сохранением ограничения.

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

Поиск с ограничением глубины может не найти решение, если оно лежит глубже, чем заданный лимит. Даже если решение существует, но оно находится за пределами ограничения, алгоритм его не обнаружит.

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

Поиск в ширину и поиск в глубину при реализации с использованием очереди отличаются тем, что в поиске в ширину используется очередь FIFO, где узлы извлекаются в порядке их добавления. В поиске в глубину используется структура данных LIFO (стек), где узлы извлекаются в обратном порядке их добавления (последний добавленный извлекается первым).

8. Почему поиск с ограничением глубины не является оптимальным?

Поиск с ограничением глубины не является оптимальным, потому что может не найти наикратчайший путь. Он может завершить поиск на более глубоком уровне, где существует более короткое решение, не достигнув его.

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

Итеративное углубление улучшает стандартный поиск с ограничением глубины, выполняя несколько поисков с увеличивающимся лимитом глубины. Это позволяет найти решение, если оно существует, даже если оно

лежит на большой глубине, и при этом избежать излишней траты времени на слишком глубокие пути.

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

Итеративное углубление становится эффективнее простого поиска в ширину, когда пространство поиска велико, а решение находится на большой глубине. В отличие от поиска в ширину, итеративное углубление не требует хранения всех узлов на каждом уровне, что экономит память.

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

Основная цель использования алгоритма поиска с ограничением глубины — это управление ресурсами, такими как время и память, путём ограничения глубины поиска. Это важно, если пространство поиска огромно, и нам нужно избежать бесконечных ветвей или избыточного расхода памяти.

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

- `cities` — граф (в словарной форме), который представляет связи между городами.
- `start` — начальная вершина (город).
- `end` — целевая вершина (город).
- `limit` — максимальная глубина поиска.

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

Значение по умолчанию параметра `limit` в функции `depth_limited_search` обычно задается в коде как фиксированное значение или передается

пользователем, если не указано, может быть установлено на уровне 3, 5 и т. д.

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

Переменная `frontier` представляет собой множество или очередь, в которой хранятся узлы для дальнейшей обработки. В алгоритме это используется для хранения текущих узлов, которые должны быть расширены или исследованы.

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

`LIFOQueue` представляет собой структуру данных, работающую по принципу "последним пришел — первым вышел" (LIFO). Это используется в алгоритме для поиска в глубину, где необходимо обрабатывать самые последние добавленные узлы.

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

Значение переменной `result` при инициализации — это список или другое контейнерное хранилище, в котором будут собраны найденные решения, пути или маршруты.

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

Цикл `while` завершится, когда очередь или стек будут пустыми, что означает, что все возможные узлы для обработки были исследованы.

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

Узел, извлекаемый с помощью `frontier.pop()`, — это последний добавленный узел, потому что структура данных — стек (LIFO).

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

Если найден узел, удовлетворяющий условию цели, то алгоритм завершает поиск и возвращает найденный путь или решение.

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

Условие `elif len(node) >= limit` проверяет, не превышает ли глубина текущего узла заданный лимит. Если условие выполняется, дальнейший поиск по этому пути обрывается.

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

Если текущий узел достигнет ограничения по глубине поиска, то дальнейшее углубление по этому пути прекращается, и алгоритм будет искать другие возможные пути с меньшей глубиной.

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

Проверка на циклы (`elif not is_cycle(node)`) в алгоритме позволяет избежать возвращения в уже посещенные узлы, что предотвращает заикливание.

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

С дочерними узлами, полученными с помощью `expand(problem, node)`, происходит следующее: они добавляются в очередь или стек для дальнейшего исследования, если они еще не были посещены.

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

Если целевой узел не был найден, функция возвращает либо `failure`, либо `cutoff`, в зависимости от ситуации.

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

- `failure` означает, что решение не было найдено.
- `cutoff` означает, что поиск был обрезан из-за достижения лимита глубины, но решение могло бы быть найдено, если бы ограничение было выше.

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

Ссылка на репозиторий:

[IUnnamedUserI/AI_4: Искусственный интеллект в профессиональной сфере. Лабораторная работа №4](#)