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 №3
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 ограничением глубины

Ссылка на репозиторий: https://github.com/Ivanuschka/4-LB

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

Порядок выполнения работы

Задание 1

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

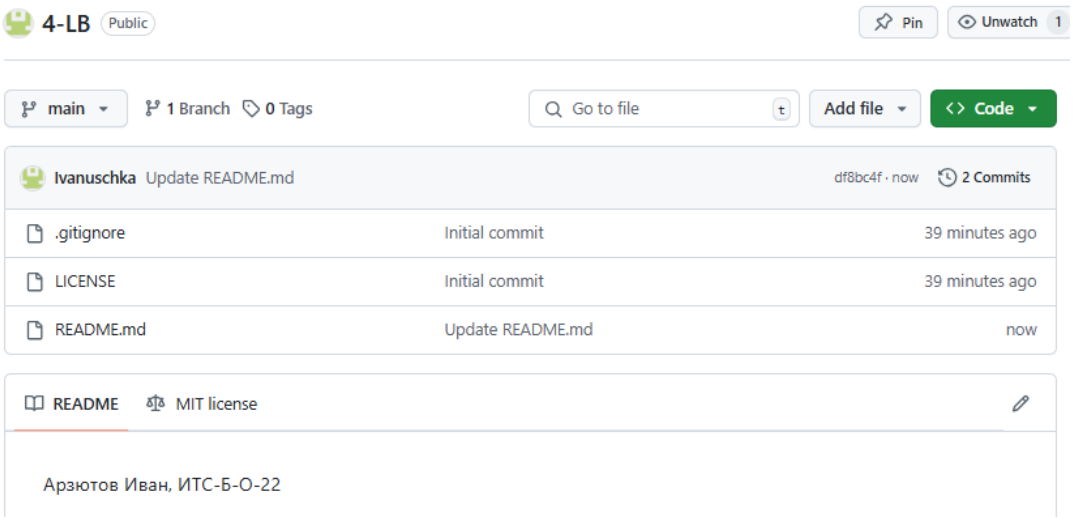

Рисунок 1. Создан новый репозиторий

Задание 2

Клонирование репозитория на свой компьютер.

```
C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа>git clone https://github.com/Ivanuschka/4-LB.git
Cloning into '4-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\4-LB

C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа\4-LB>
```

Рисунок 2. Клонирование репозитория

Задание 3

Проработка примера лабораторной работы

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

Рисунок 3. Выполнение примера

Задание 4

Создание модулей для примера

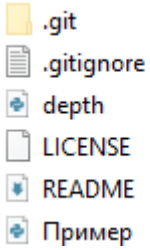

Рисунок 4. Модули в папке репозитория

Задание 5

Для построенного графа написать 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.

```python
import numpy as np
import matplotlib.pyplot as plt

def dls(graph, current, target, visited, path, current_distance, depth, max_depth, result):
    """Рекурсивный поиск в глубину с ограничением"""
    if depth > max_depth:
        return False  # Достигнут лимит глубины

    visited.add(current)
    path.append(current)

    if current == target:
        if current_distance < result["min_distance"]:
            result["min_distance"] = current_distance
            result["min_path"] = path.copy()
            result["found_depth"] = depth
        visited.remove(current)
        path.pop()
        return True

    for neighbor, distance in graph[current]:
        if neighbor not in visited:
            if dls(graph, neighbor, target, visited, path, current_distance + distance, depth + 1, max_depth, result):
                return True

    visited.remove(current)
    path.pop()
    return False

def find_shortest_path_dls(graph, start, end, max_depth):
    """Функция для поиска кратчайшего пути с ограничением глубины"""
    result = {"min_distance": float('inf'), "min_path": [], "found_depth": -1}

    for depth in range(max_depth + 1):
        if dls(graph, start, end, set(), [], 0, 0, depth, result):
            return result["min_path"], result["min_distance"], result["found_depth"]

    return [], float('inf'), -1  # Если путь не найден

# Определяем граф (города и расстояния)
graph = {
    'Город A': [('Город B', 15), ('Город J', 14)],
    'Город B': [('Город A', 15), ('Город D', 12), ('Город H', 25)],
    'Город C': [('Город A', 10), ('Город D', 8), ('Город E', 20)],
    'Город D': [('Город C', 8), ('Город B', 12), ('Город E', 5), ('Город F', 5), ('Город G', 22)],
    'Город E': [('Город C', 20), ('Город D', 5), ('Город F', 18)],
    'Город F': [('Город E', 18), ('Город D', 5), ('Город G', 7)],
    'Город G': [('Город D', 22), ('Город F', 7), ('Город H', 10)],
    'Город H': [('Город G', 10), ('Город B', 25), ('Город I', 6)],
    'Город I': [('Город H', 6), ('Город J', 8)],
    'Город J': [('Город I', 8), ('Город A', 14)]
}

start_city = 'Город E'
end_city = 'Город I'
max_depth = 10  # Ограничение глубины поиска

shortest_path, shortest_distance, found_depth = find_shortest_path_dls(graph, start_city, end_city, max_depth)

if shortest_path:
    print(f'🔹 Кратчайший путь из {start_city} в {end_city}: {" -> ".join(shortest_path)}')
    print(f'🔹 Общая длина пути: {shortest_distance} км')
    print(f'🔹 Решение найдено на глубине {found_depth}')
else:
    print(f'❌ Путь из {start_city} в {end_city} не найден при глубине {max_depth}')
```

Рисунок 5. Код для определения кротчайшего пути с ограниченным глубинным поиском

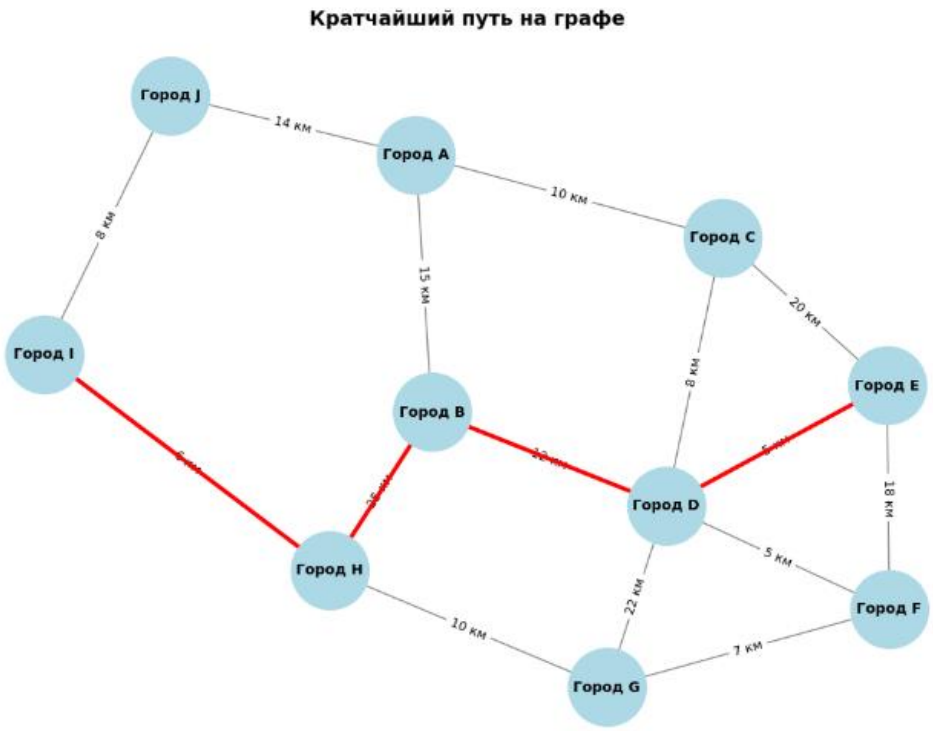

Рисунок 6. Визуализация кратчайшего пути при поиске с ограниченной глубиной

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

Ответы на контрольные вопросы

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

Поиск с ограничением глубины – это метод поиска, в котором алгоритм ограничивает глубину обхода дерева или графа. Он решает проблему бесконечных ветвей, устанавливая максимальное значение глубины, на которое алгоритм будет исследовать узлы. Если это ограничение достигнуто, дальнейшее исследование прекращается, предотвращая зацикливание.

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

Основная цель ограничения глубины – избежать излишнего расходования ресурсов при поиске, обеспечивая при этом возможность исследования более глубоких уровней дерева поиска, если они потенциально ведут к решению.

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

Поиск в глубину продолжает обходить узлы, пока не найдет решение или не достигнет конца дерева, тогда как поиск с ограничением глубины прекращает поиск, если достигнута заранее заданная глубина, что ограничивает количество исследуемых узлов.

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

Проверка глубины узла позволяет алгоритму определить, достигнуто ли установленное ограничение, и соответственно решить, продолжать поиск дальше или обрезать его.

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

Возврат «обрезания» указывает на то, что поиск достиг установленного лимита глубины и больше не будет продолжаться. Это позволяет алгоритму понимать, что данная ветвь не приведёт к решению.

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 целевое состояние находится на глубине, превышающей установленный предел, что может произойти, если глубина решения не предсказуема заранее.

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

Поиск в ширину использует очередь (FIFO), помещая узлы в один конец и извлекая из другого, что обеспечивает исследование всех узлов на текущем уровне перед переходом к следующему. Поиск в глубину использует стек (LIFO), помещая новые узлы на вершину и извлекая с неё, что значит углубление до конца в одной ветке перед исследованием других.

8. Почему поиск с ограничением глубины не является оптимальным?

Поиск с ограничением глубины не является оптимальным, потому что он может не рассматривать более глубокие уровни, которые могут привести к более коротким и эффективным решениям из-за ограничения глубины.

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

Итеративное углубление сочетает преимущества поиска в глубину и поиска с ограничением глубины. Он выполняет поиск с увеличивающимся пределом глубины, начиная с нуля и увеличивая его, что позволяет находить

решение, если оно существует, даже если оно находится глубже, чем изначально предполагалось.

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

Итеративное углубление становится более эффективным, когда память ограничена, поскольку оно не требует хранения всех уровней узлов, как это делает поиск в ширину. Это позволяет избежать потребления ресурсов за счёт сохранения только текущего уровня узлов.

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

Основная цель использования алгоритма поиска с ограничением глубины заключается в контроле глубины поиска для избежания чрезмерного расхода ресурсов и управления бесконечными ветвями.

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

Функция depth_limited_search принимает параметры, такие как problem (задача) и limit (максимальная глубина). problem определяет структуру задачи и её состояние, а limit ограничивает глубину обхода.

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

Значение по умолчанию для параметра limit чаще всего устанавливается в ноль или в некое небольшое значение, что позволяет избежать излишних ресурсов в случае отсутствия явных указаний на глубину.

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

Переменная frontier представляет собой структуру данных, в которой хранятся узлы, ожидающие обработки. Она управляет узлами, которые будут исследованы на следующих этапах алгоритма.

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

LIFOQueue – это структура данных, работающая по принципу «последний пришёл – первый вышел» (Last In, First Out). Она используется в алгоритме для реализации поиска в глубину.

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

Значение переменной result обычно инициализируется как None, указывая на то, что решение ещё не найдено.

17. Какое условие завершает цикл while в алгоритме поиска?

Цикл while завершается, когда frontier пуст, что означает, что больше нет узлов для исследования.

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

С помощью frontier.pop() извлекается последний добавленный узел, так как алгоритм работает по принципу LIFO, что позволяет исследовать узлы в порядке их добавления.

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

Если найден узел, удовлетворяющий условию цели, алгоритм возвращает его как решение.

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

Условие `elif len(node) >= limit` проверяет, достиг ли узел предельной глубины. Если это так, то узел не будет дальше обрабатываться и алгоритм вернётся к предыдущему узлу.

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

Если текущий узел достигает ограничения по глубине, алгоритм не исследует его дочерние узлы и возвращает «обрезание», указывая, что дальше не имеет смысла продолжать поиск на этой ветви.

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

Проверка elif not is_cycle(node) определяет, является ли узел циклом, и, при необходимости, предотвращает повторное исследование узлов, чтобы избежать зацикливания.

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

Дочерние узлы, полученные с помощью expand, добавляются в frontier для дальнейшего исследования.

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

Если целевой узел не был найден, функция обычно возвращает значение, указывающее на неудачу, например failure или cutoff.

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

Результат failure указывает на то, что решение не существует, в то время как cutoff означает, что достигнута предельная глубина, и дальнейший поиск не возможен, но решение могло бы быть найдено на больших глубинах.